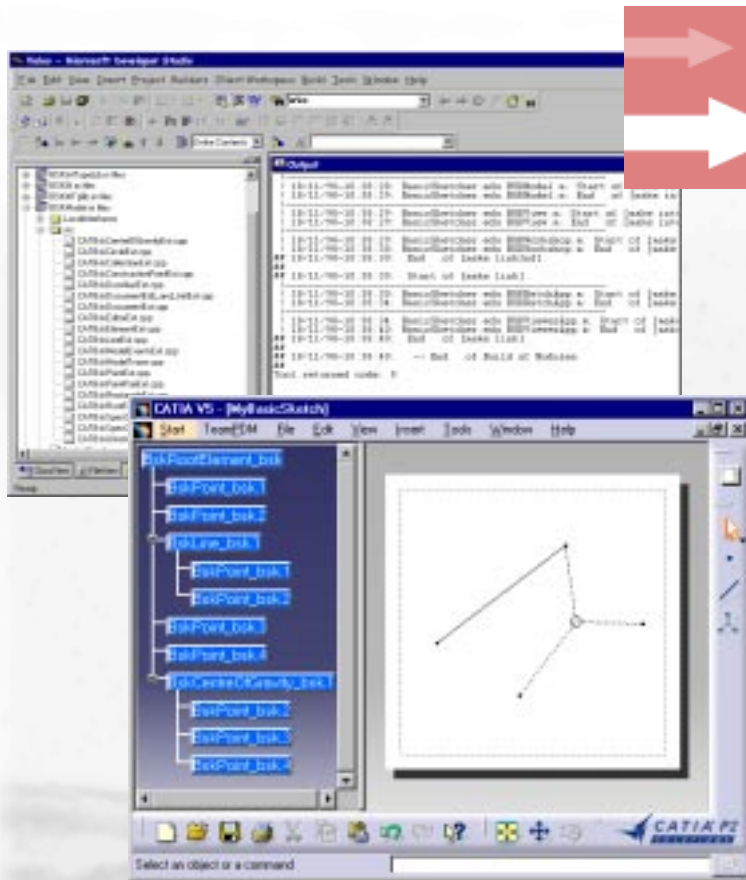


CATIA Training Foils



CAA V5 For CATIA Foundations

Version 5 Release 7
July 2001

FOR-CAT-E-CAA-F-V5R7

CAA V5 For CATIA Foundations

Objectives of the Course

In this course you will learn the CAA V5 development platform, know the foundation components, and understand the architecture of a CATIA V5 application following the Model/View/Controller design pattern.

Targeted audience

C++ Programmers who intend to develop CAA Applications (interactive or batch)



Prerequisites:

CATIA V5 user interface principles (Mandatory)

C++ industrial programming practice (Mandatory)

COM (Microsoft Object Model) notions (Nice to have)

Microsoft Developer Studio practice (Nice to have)

Table of Contents

1. Native CATIA V5 Openness Positioning

Introduction

Openness tools

Some Case studies

Recommendations

2. CAA V5 Development Environment

A Component Architecture

Workspaces and Frameworks

Compilation Tools

MSDev Integration

Other Tools

CAA V5 Encyclopedia

Programming Rules

3. CAA V5 Object Modeler

Why a New Object Modeler?

The Interface / Implementation Pattern

Between Interfaces and implementations

Object Life Cycle Management

Extension Mechanism

Late Typing

Table of Contents – Continued

- 4. **CAA V5 ObjectSpecsModeler**
 - Introduction to Specification Modeling**
 - Object Specs Modeler Objectives**
 - About Features**
 - Programming Tasks**
 - About Features Extensions**
 - Programming Tasks**
- 5. **CAA V5 Visualization**
 - Framework Objectives**
 - Manage the Presentation**
 - Model / View / Controller Architecture**
 - Visualization Interfaces**
- 6. **CAA V5 ApplicationFrame**
- 7. **CAA V5 DialogEngine**
 - Objectives of CATIA V5 DialogEngine**
 - Main notions**
 - How to define a new interactive command**

Table of Contents – Continued

8. CAA V5 Dialog

- Framework objectives
- Building graphic user interfaces
- Retrieving user inputs
- The Dialog builder

9. CAA V5 Administration

- Packaging
- Licensing
- Software Prerequisites
- Delivering a CAA build Application

You will learn what are the different techniques and tools available in CATIA V5 to customize it and when to use which one.


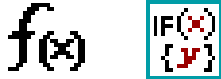
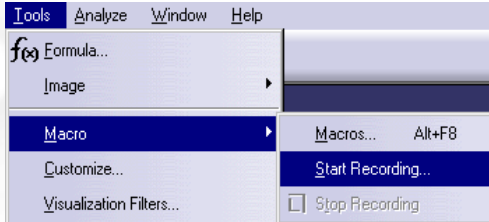


- Introduction
- Openness tools
- Some Case studies
- Recommendations

Copyright DASSAULT SYSTEMES 2000

6


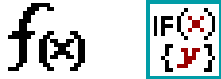
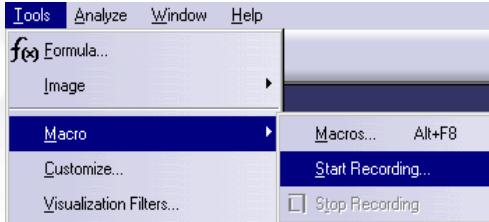


You will learn what are the different techniques and tools available in CATIA V5 to customize it and when to use which one.

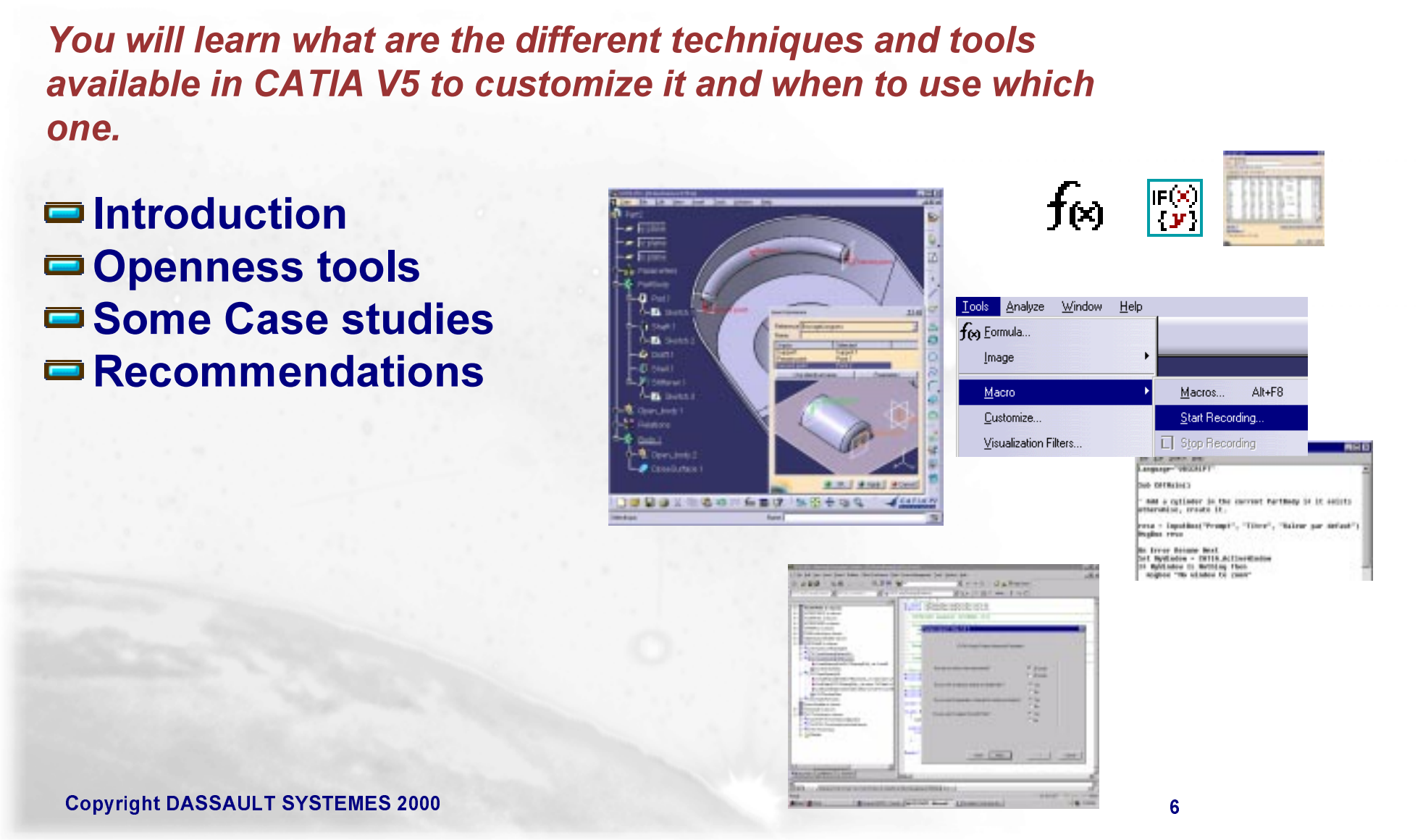
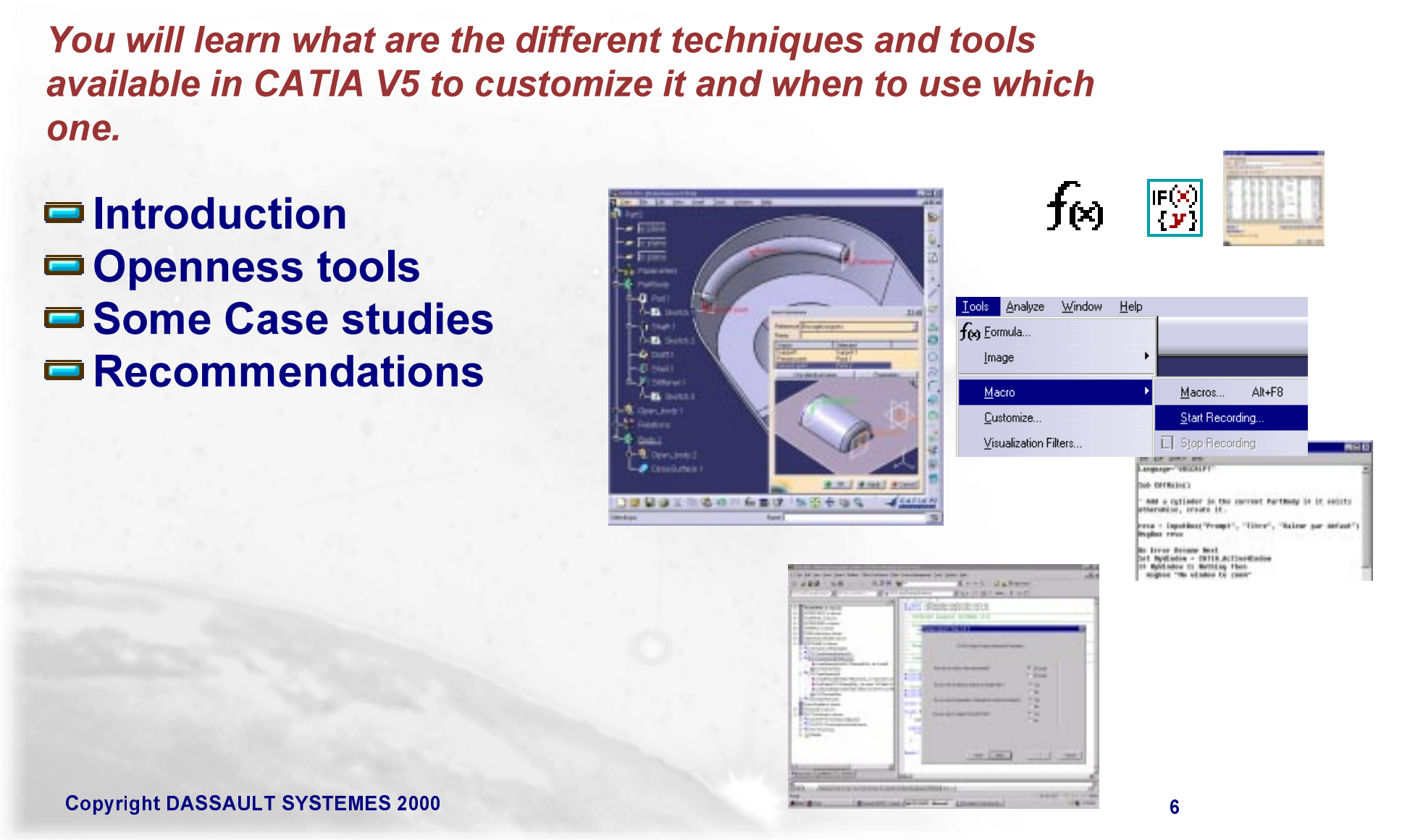
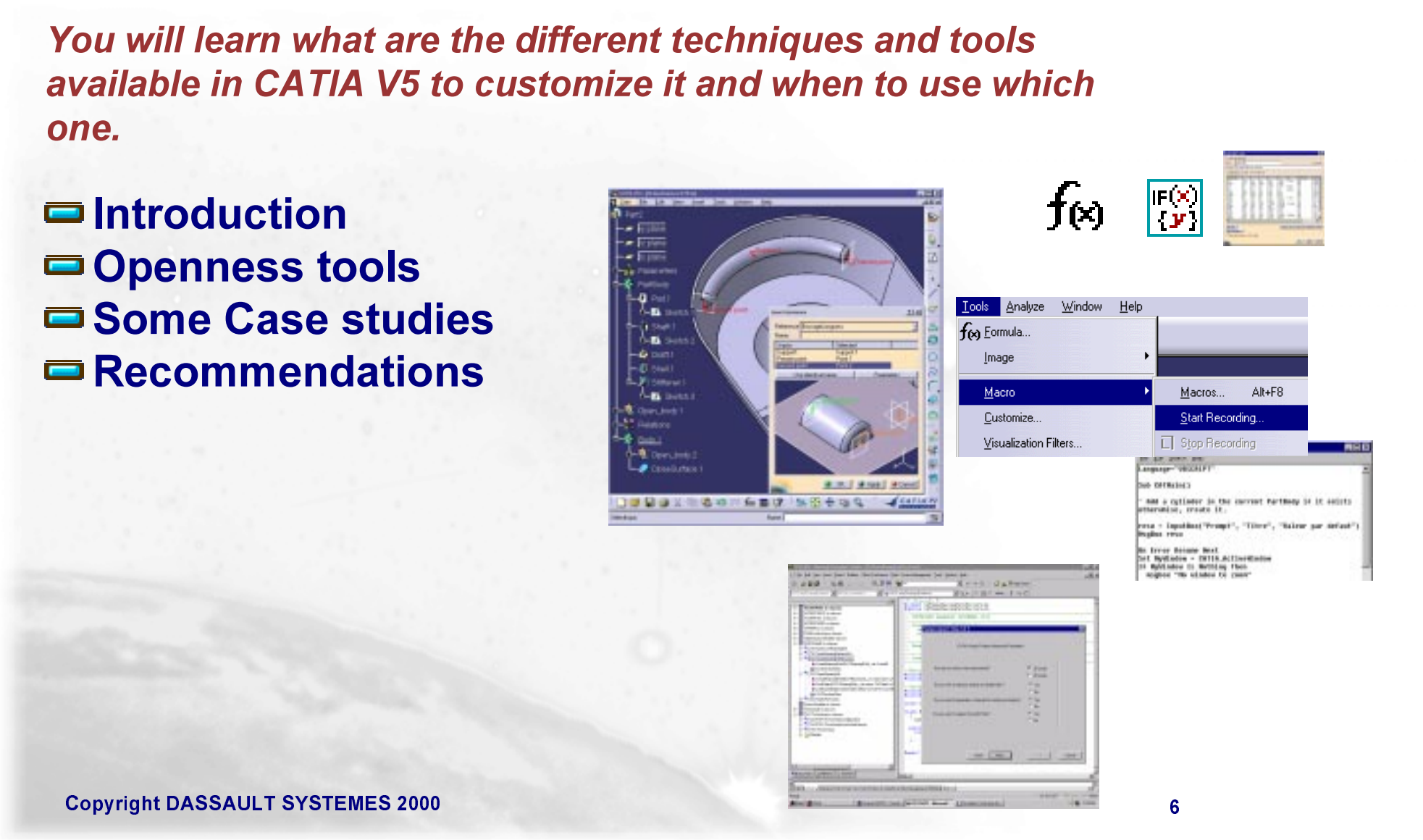
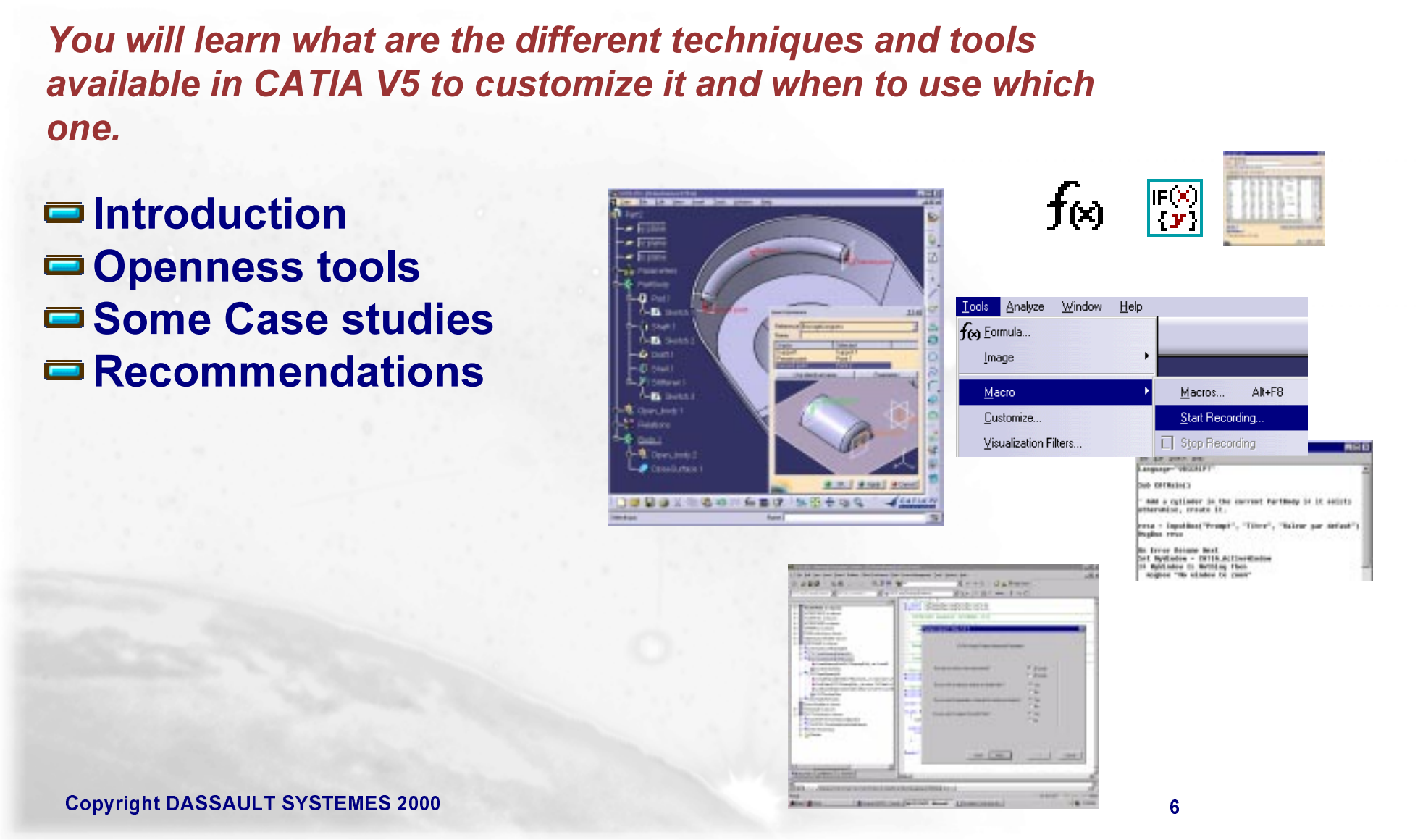
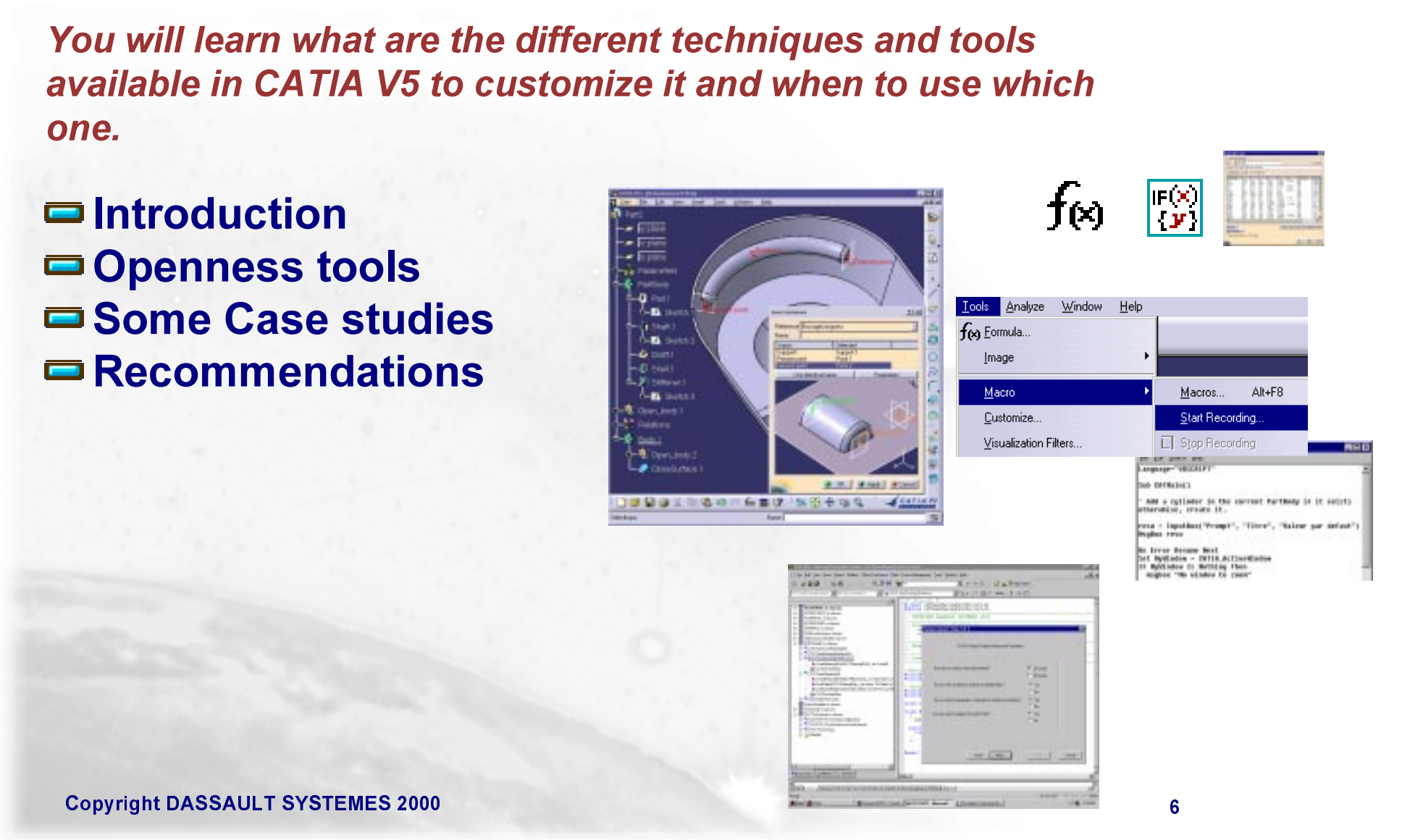
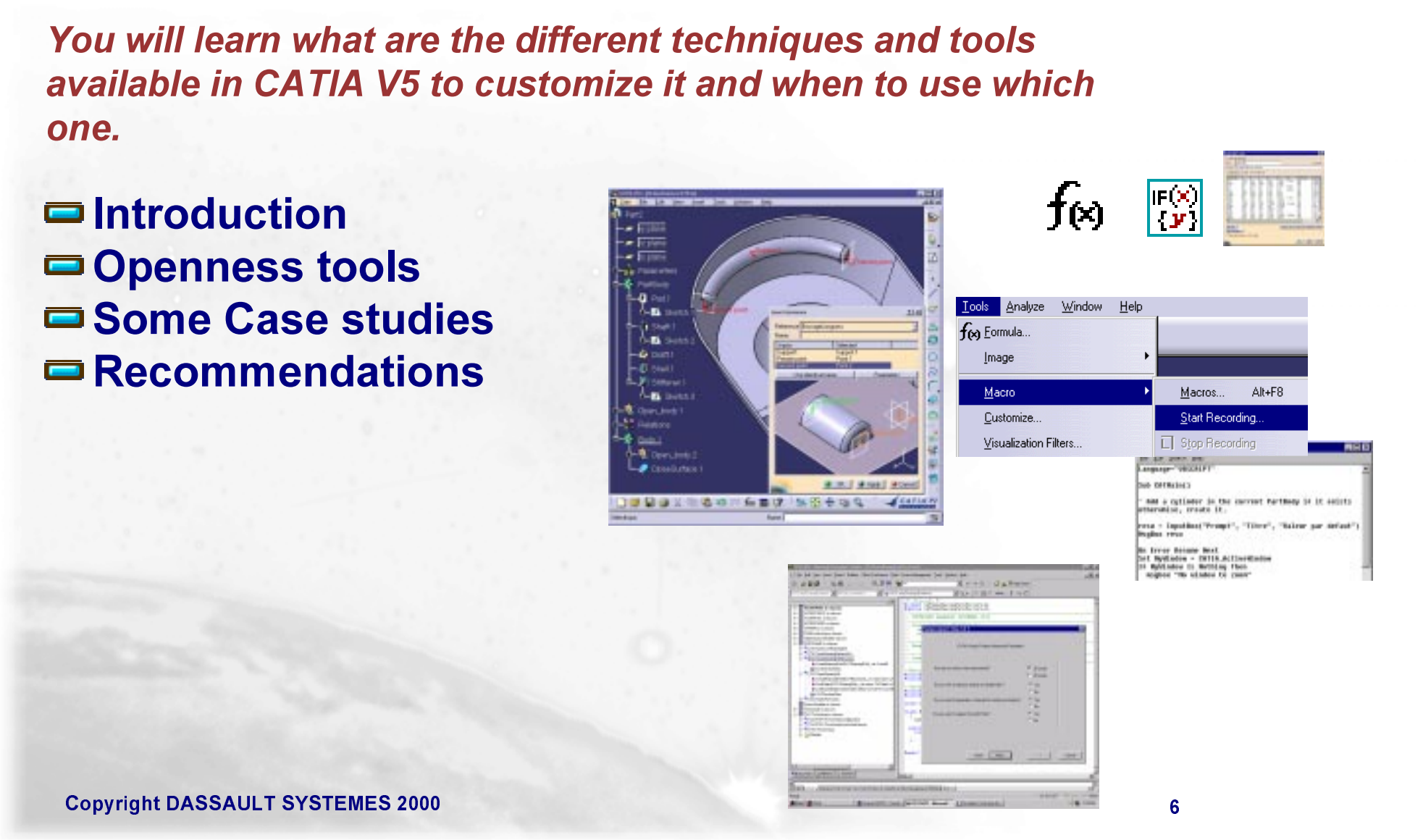
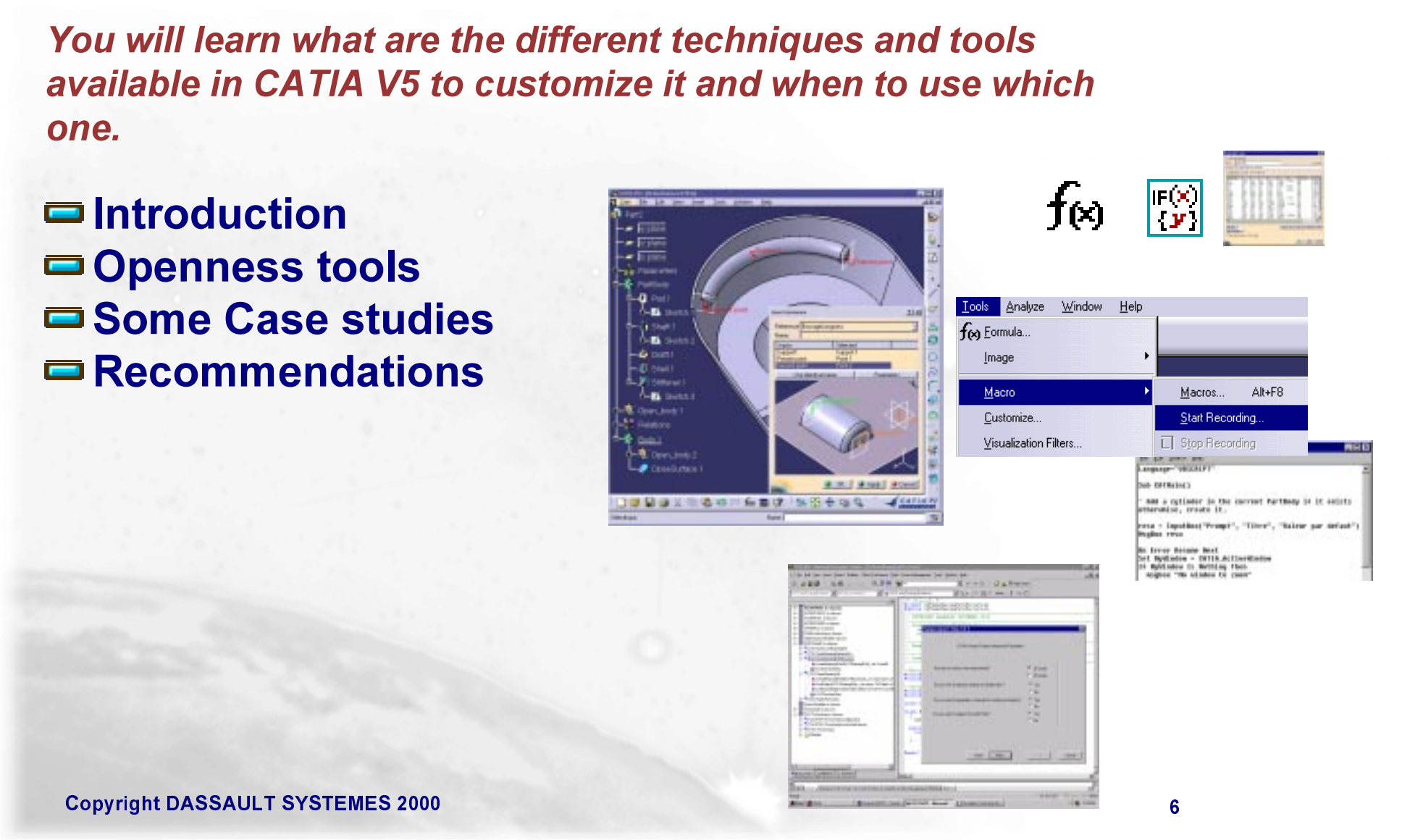
- Introduction
- Openness tools
- Some Case studies
- Recommendations

Copyright DASSAULT SYSTEMES 2000

6

- You will learn what are the different techniques and tools available in CATIA V5 to customize it and when to use which one.*
- Introduction
 - Openness tools
 - Some Case studies
 - Recommendations
- 




- Copyright DASSAULT SYSTEMES 2000
- 6



Introduction



Many openness capabilities available in CATIA V5

Standard format import/export (V5R1)

Macros using the Automation API (V5R1)

Knowledgeware (V5R1)

Interactive User Defined Feature (V5R7)

CAA V5 C++ & Java API (V5R6)



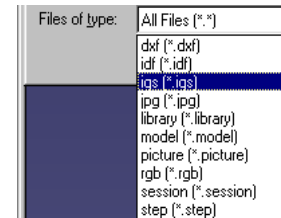
***What are these tools provided for ?
When to use what ?***

Different Levels



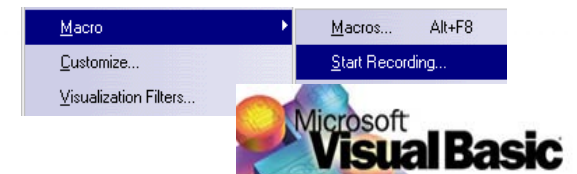
Multi-CAX, Multi-PDM, Standard Format Import/Export

«by conversion» customization



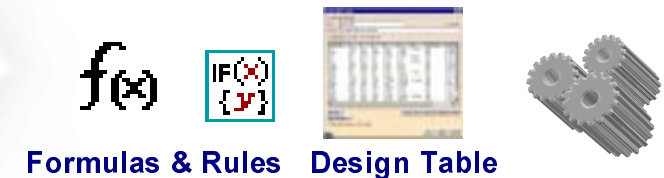
Automation Components, Journaling, VisualBasic, JavaScript/HTML

«interactive» customization



Knowledgeware

«reactive, rule-based, by goal» customization



Formulas & Rules Design Table



Interactive User Defined Feature

«by composition» customization

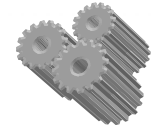
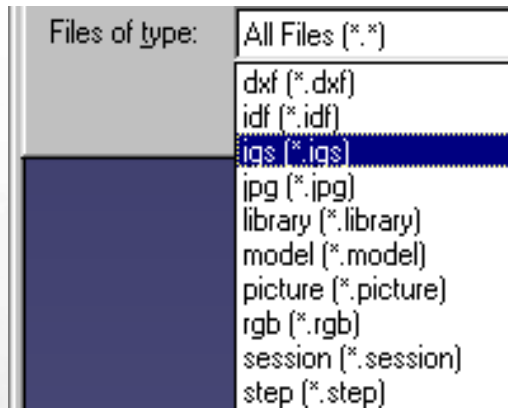


Java and C++ Components

«component-based programming» customization



Openness Tools Highlights



Formulas & Rules Design Table



CATIA V5 Automation API

*Automation API to be used in macros written in an interpreted language:
Visual Basic Script or Java Script*

**CATIA Automation: end user view
on the CATIA data model**

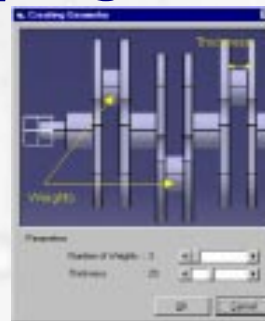
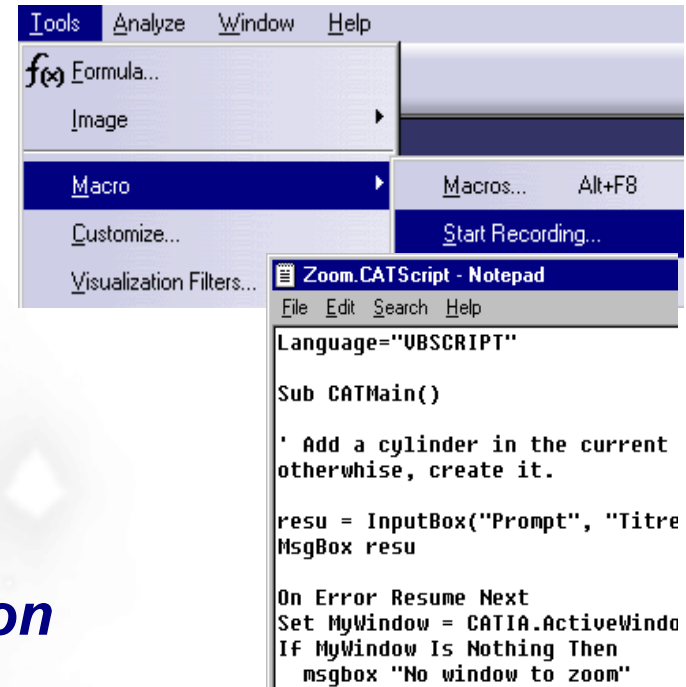
Rely on standard languages

Journaling: interactive programming

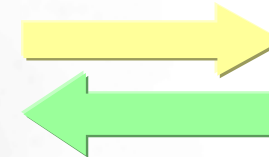
UNIX & WINDOWS NT

Basic Script (Summit)

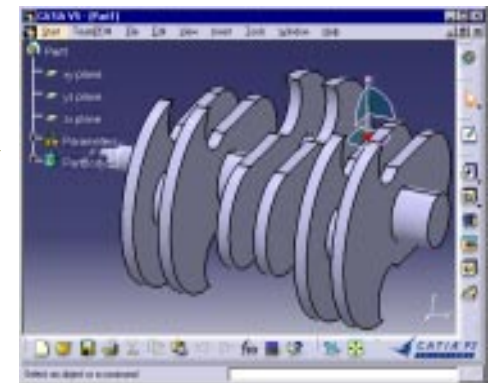
**Communication with any OLE
compliant application (NT)**



VB application
1. Launch CATIA
2. Generate data



VB application
3. Retrieve information
from CATIA



CATIA V5 Knowledgeware

*A productivity environment for specification reuse:
as simple as formula to corporate knowledge base rules*



***Fully integrated platform in V5
in order to leverage processes***

*Imbedded Knowledge
that participates to
the design definition*

Knowledge
Advisor

Rules
Checks
Inspector
Behavior

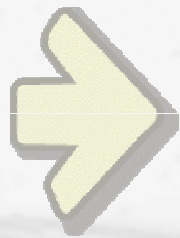


***Interactive knowledge capture
through associative and persistent
specification definition and
predefined easy to use services***

*Rule based
processes*

Knowledge
Expert

Rules
Checks
Rule Based
Reports



***Driven management and reuse:
from functions, specifications to
components and systems***



***Allow customization and external
sources integration***

Design by goal

Product
Engineering
Optimizer

Optimization goal
Criteria
Solutions



Interactive User Defined Feature

*Define interactively new data types
by aggregating existing features*



Interactive data definition



***Collect existing specifications,
specify inputs and create a «IUDF»***

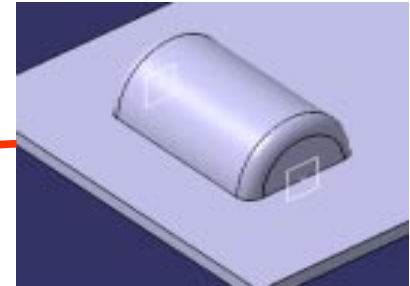


***«IUDF» to be saved in a CATPart
document and referenced in a catalog***

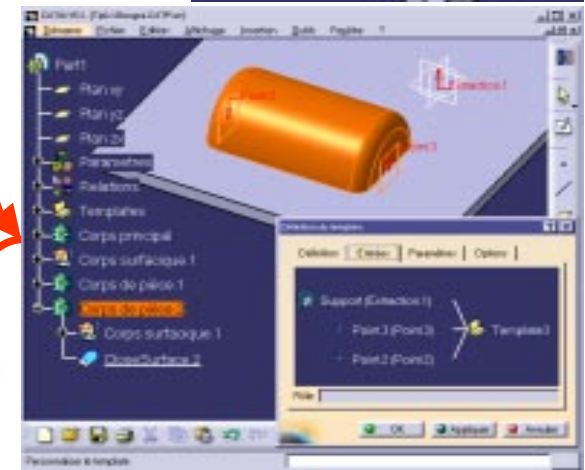


Interactive easy reuse

Reference
geometry



1. Definition



2. Stored
in catalog



3. Instantiated
in context

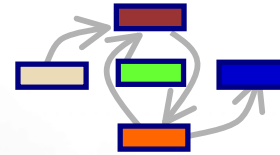


CAA V5 C++ and Java API

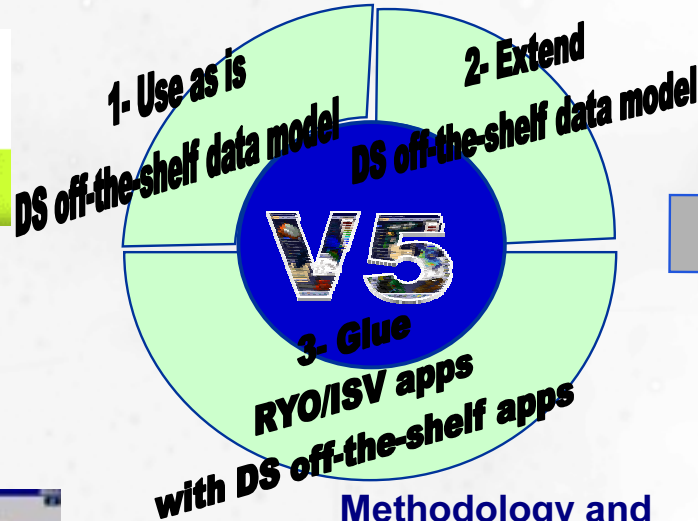
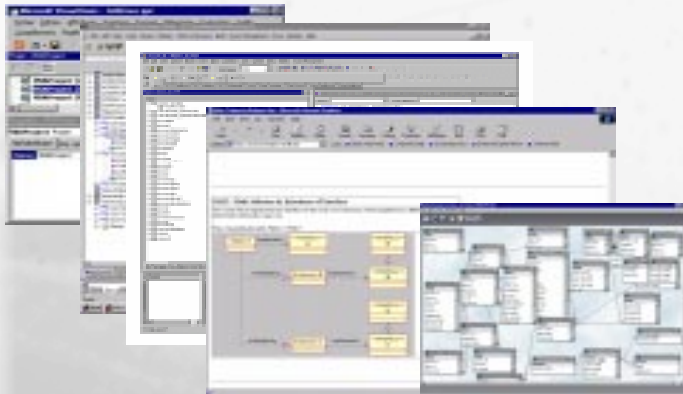
Based on extensions
of standards tools

Teamwork
development

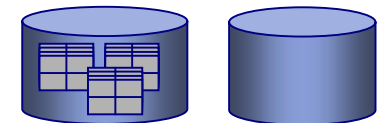
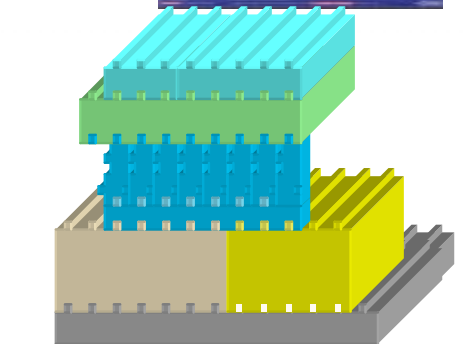
Full software development
lifecycle coverage



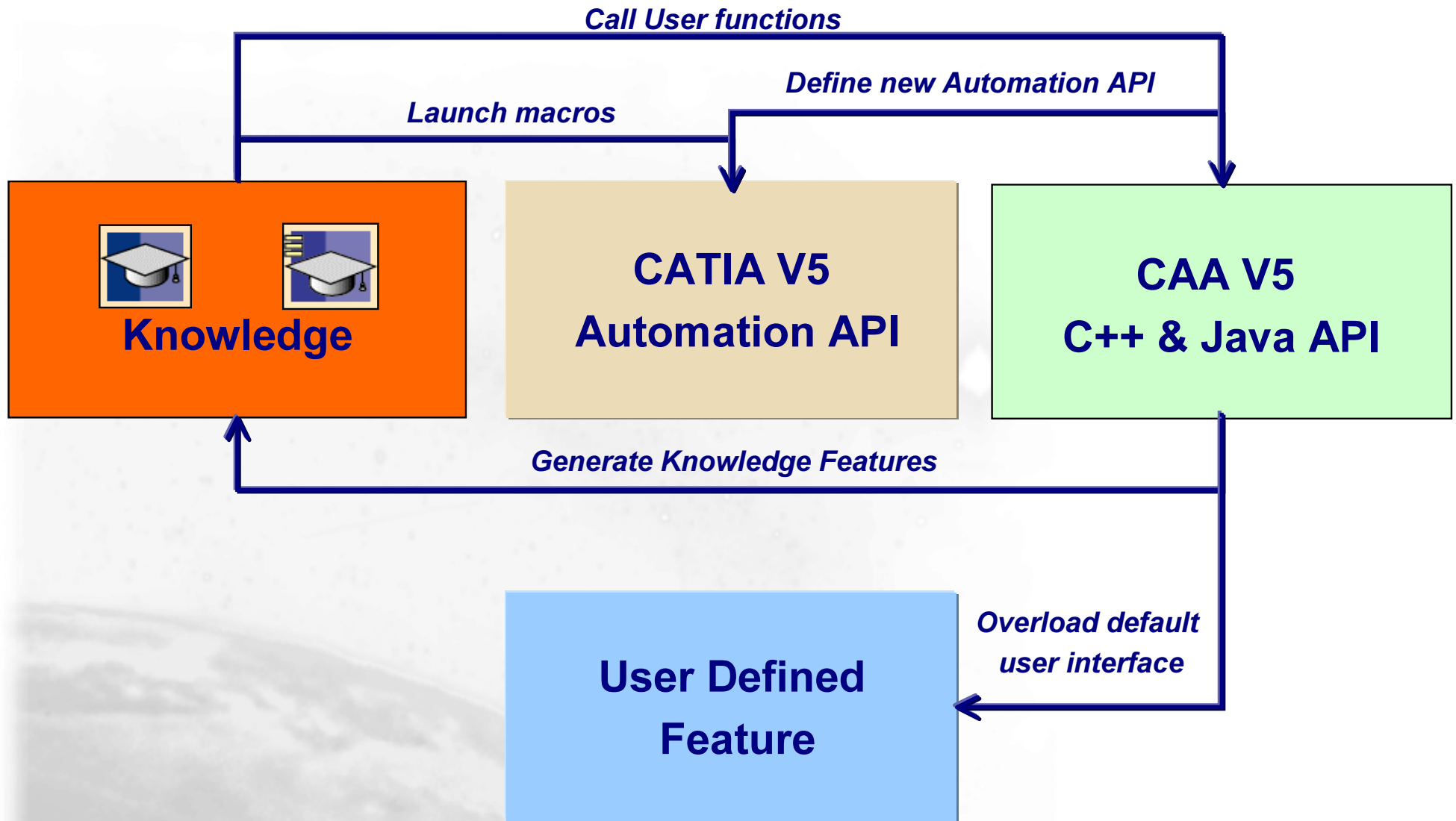
Interactive
Development



Methodology and
programming guides



Tools Collaboration



Some Case Studies

Use as is

-  Capture an interactive process
-  Define some new user interface
-  Check corporate rules

Extend

-  Enrich CATIA V5 data model
-  Create or modify data behavior

Glue

-  Glue with external systems

Use as is: Capture interactive process

ex: perform in one shot a sequence of operations on DS objects



CATIA V5
Automation API



CAA V5
C++ & Java API

Use as is: Define some new user interface

ex: ask for user parameter inputs in a panel



CATIA V5 Automation API

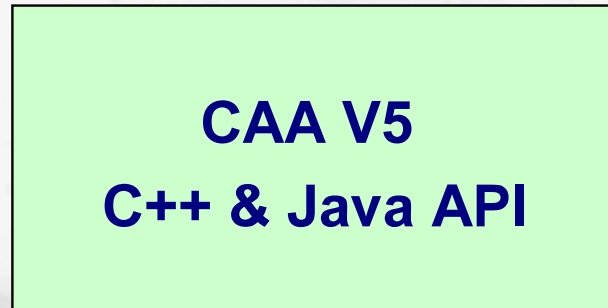
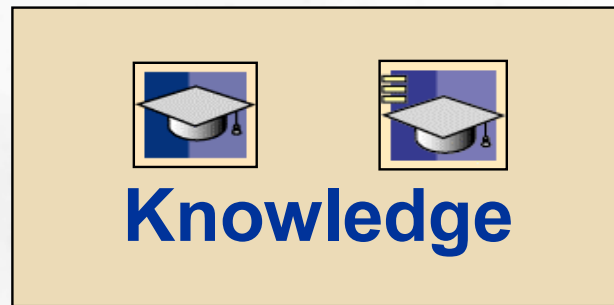
- ▶ *Very simple panels with “InputDialog” and “MsgBox”*
- ▶ *More sophisticated panels by defining an ActiveX in Visual Basic. (NT only)*
- ▶ *Associate an icon to a macro and plug it in the CATIA frame*
- ▶ *Selection capabilities (V5R6)*



CAA V5 C++ & Java API

- ▶ *Sophisticated interactive commands*
- ▶ *Command grouping in workbench*
- ▶ *State of the art panels*
- ▶ *Undo/Redo*

Use as is: Check corporate rules



Extend: Enrich CATIA V5 data model

ex: define a new feature to be seen afterwards like any other DS feature



**User Defined
Feature**

- *Atomic DS features exist.*
- *No code to be written*



**CAA V5
C++ & Java API**

Extension: Create or modify data behavior

ex: extend the behavior of an existing object



CAA V5
C++ & Java API

Glue: Communication with external systems

ex: generate new data in V5 documents or retrieve information



**Import/Export
format**



**CATIA V5
Automation API**

- ▶ *CATIA is an OLE Automation server.*
- ▶ *WINDOWS NT only*



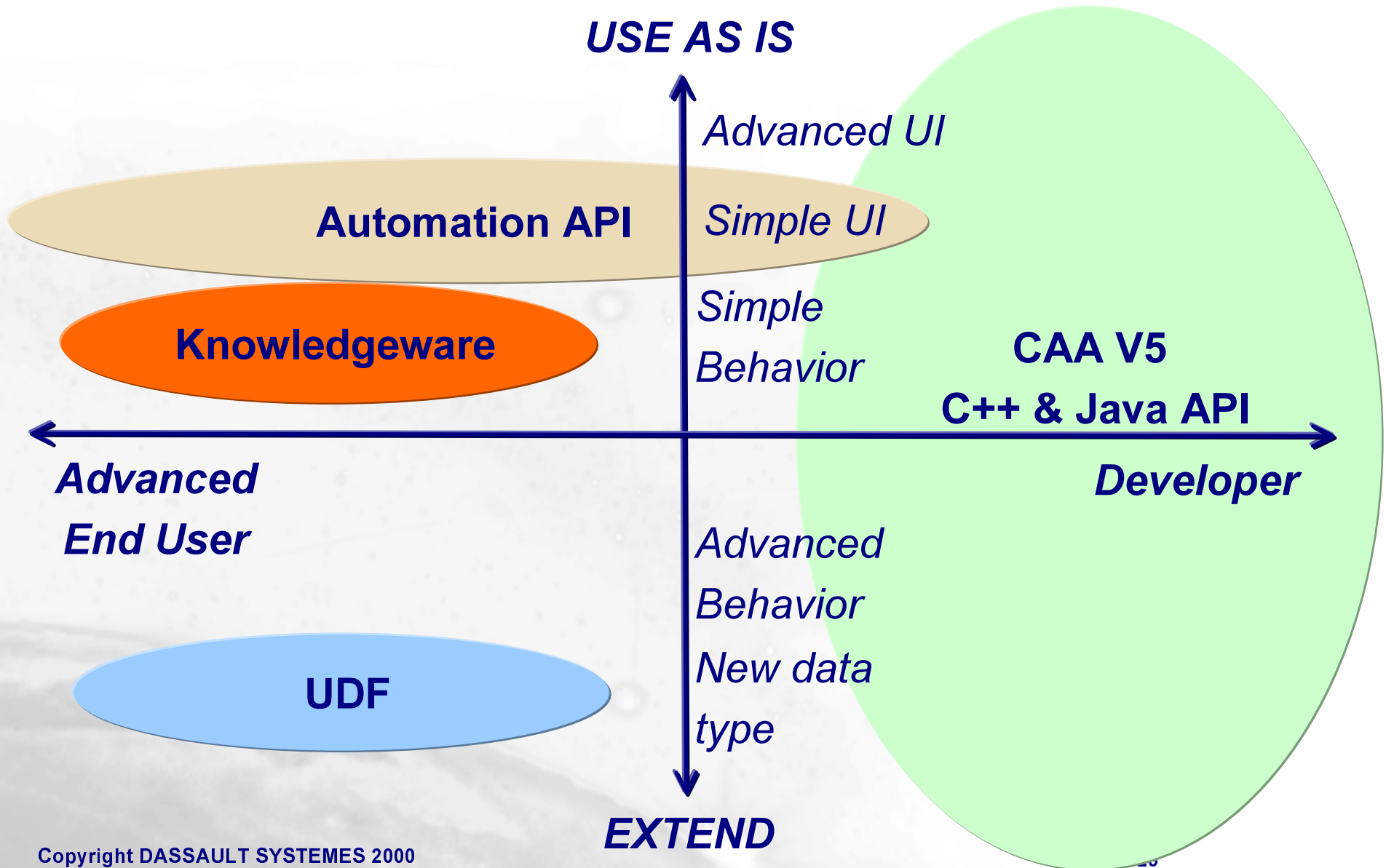
**CAA V5
C++ & Java API**

- ▶ *The Backbone proposes an asynchronous communication.*
- ▶ *Encapsulate the underlying technologies*

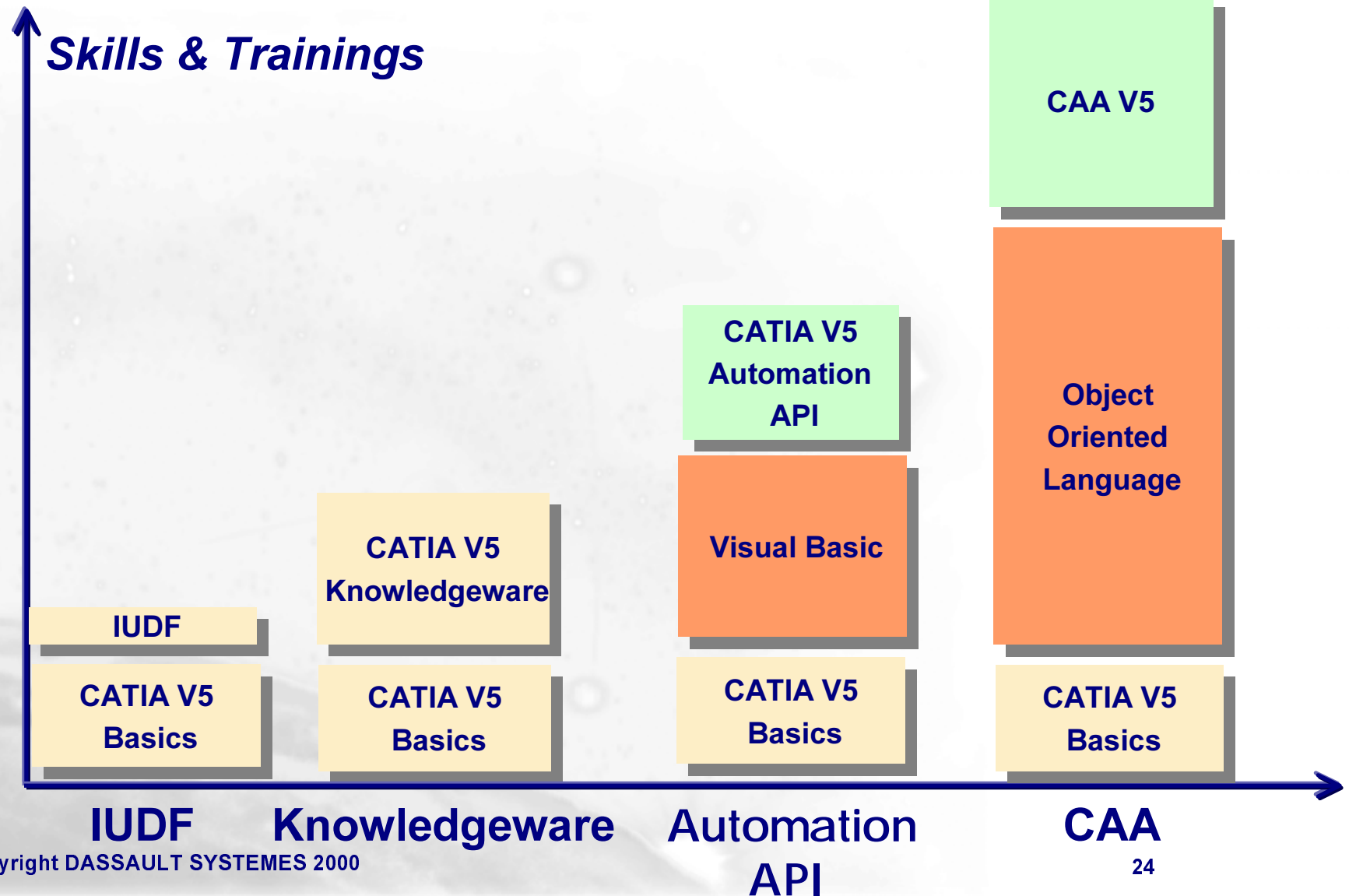
Recommendations

- ▣ Positioning according to capabilities
- ▣ Positioning according to required skills & training

Positioning according to capabilities



Positioning according to Required Skills & Trainings



To Sum Up ...

In this lesson you have seen...

- ***A range of openness tools***
 - Adapted to different targets***
 - From simple to advanced developments***
 - From end-user to professional programmer***
- ***Whatever your needs, you benefit from an appropriate tool.***

CAA V5 Development Environment

You will learn about the directory tree structure, the specific tools developed and plugged in Microsoft Developer Studio, and how to find information in the CAA V5 Encyclopedia.

- **A Component Architecture**
 - *Workspaces and Frameworks*
- **Compilation Tools**
 - *mkmk*
- **Microsoft Developer Studio Integration**
- **Other Tools**
- **CAA V5 Encyclopedia**
- **Programming Rules**

CAA V5 Development Environment Objectives



Tools and Methods for an OO programming environment



Support the V5 Architecture



Support large teams of developers working concurrently in different sites



Help making better quality software in a faster way



Capture and enforce company processes

CAA V5 Characteristics



Common development platform for all the Dassault Systemes product lines

CATIA / ENOVIA / DELMIA

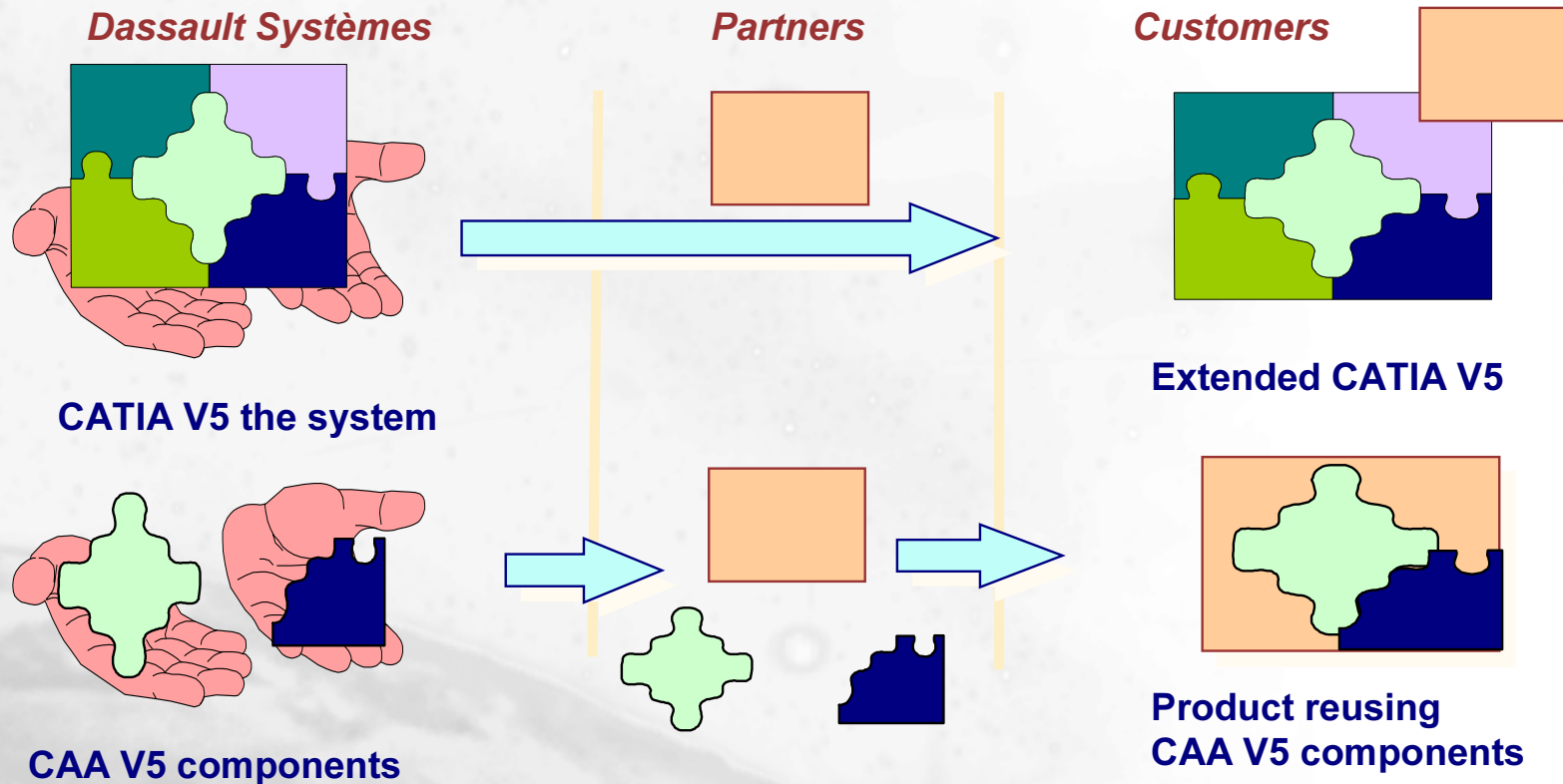


Code written on top of CAA V5 is the same on NT and UNIX

Component Application Architecture



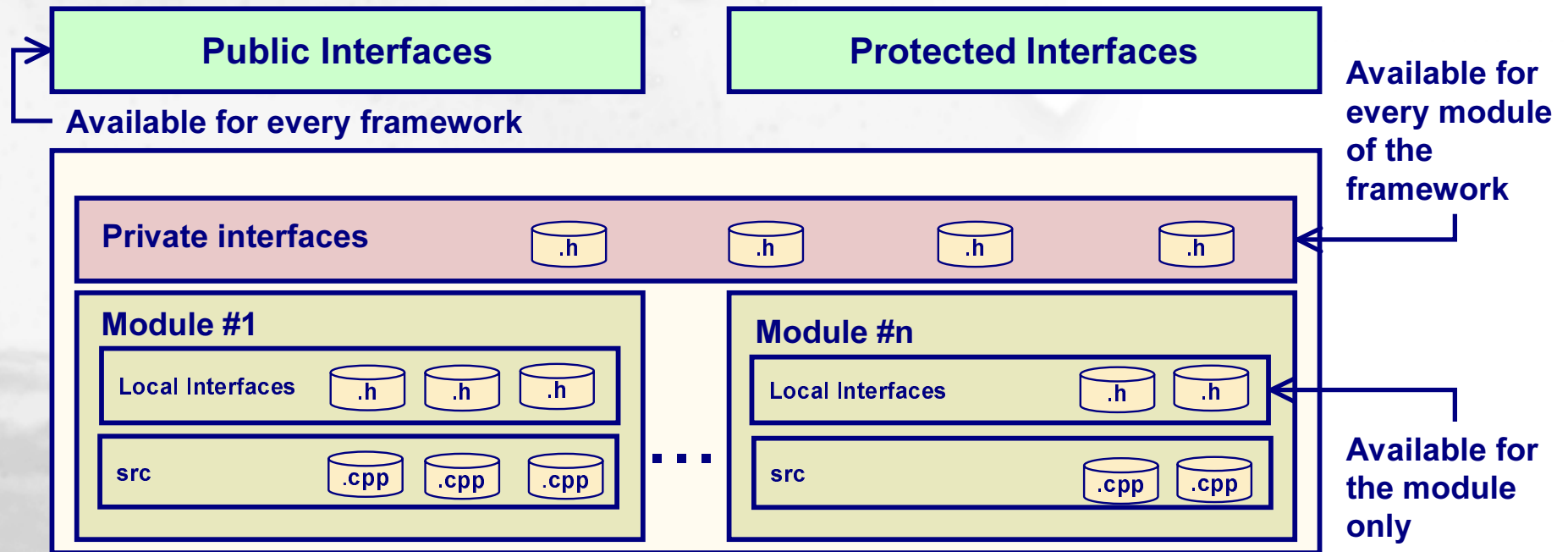
CAA V5 can be used to implement new products



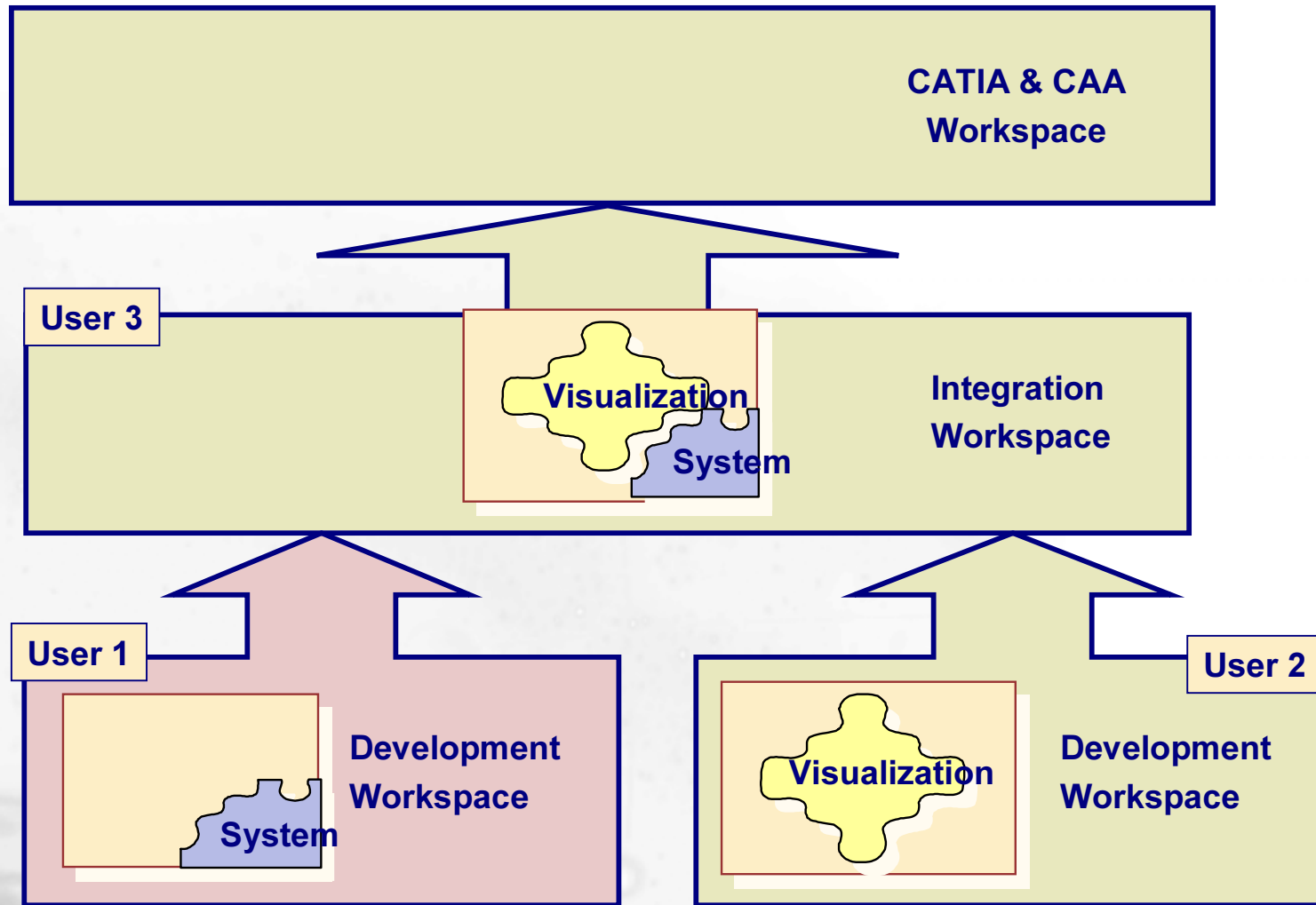
CAA V5 Framework



Group of interoperating objects with built-in capabilities delivered as a complete resource to client applications

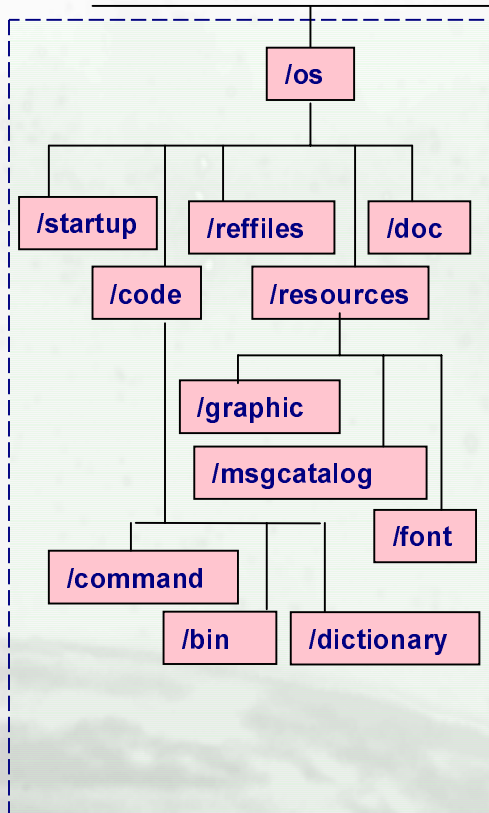


CAA V5 Pre-Requisite Workspaces

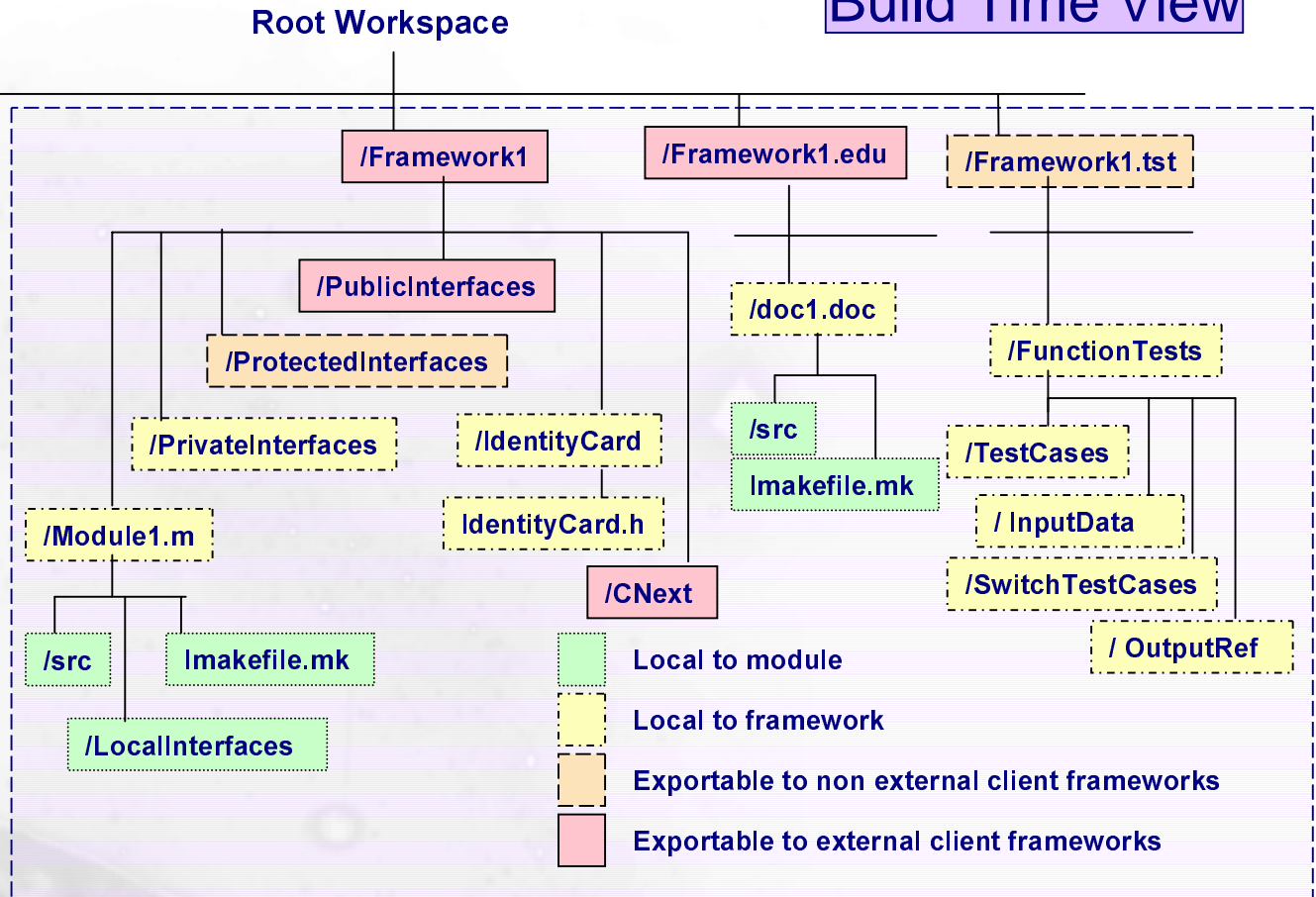


CAA V5 File Tree

Run Time View



Build Time View



Framework Identity Card



The IdentityCard defines the pre-requisite frameworks to build and use a framework.

One identity card per framework.

If no pre-requisite framework, define an empty IdentityCard.

This file is used by our building tool to limit the header file search to the corresponding Interfaces directories of the pre-requisite frameworks.

IdentityCard.h

This framework uses only headers defined in the PublicInterfaces or ProtectedInterfaces directory of the System and ObjectModelerBase frameworks

```
AddPrereqComponent("System",Protected);  
AddPrereqComponent("ObjectModelerBase", Protected);
```

Manage the CAA V5 Tool Level: TCK



The Tool Configuration Key manages several levels of the CAA V5 RADE tools.



*To set up the tck environment:
tck_init*



*To list the different levels available
tck_list*



*To set up a specific tool level
tck_profile LevelNameYouWantToUse*

Define your prerequisite workspaces: mkGetPreq



mkGetPreq -p PrerequisiteWorkspace1

This enables you to define where the prerequisite resources are located

build time: header files

run time: shared libraries, resource files, ...

This command must be launched in a window where the CAA V5 environment has been set and the current directory is your workspace

Build your executables: mkmk



A unique DS tool built on top of the standard compilers that works in the same way on UNIX and Windows NT:

***Compile Fortran, C, C++, IDL, Express, CIRCE, ...
Link-edit***



It uses the Imakefile.mk file that must be defined for every module.

mkmk: The Imakefile.mk

Imakefile.mk

```
BUILT_OBJECT_TYPE=SHARED LIBRARY
```

Define the module type

Define the build options common to all the OS

```
OS = COMMON
```

```
WIZARD_LINK_MODULES = \
```

Specific keyword used by the wizards

```
JS0GROUP JS0FM CATApplicationFrame
```

The continuation character is “\”

```
LINK_WITH = $(WIZARD_LINK_MODULES) \  
            CATDialogEngine
```

Defines the shared libraries that
resolve the symbols you use

```
OS = AIX
```

Define the build options specific to a given OS if necessary

```
SYS_INCPATH =
```

```
SYS_LIBS = -IXm -IXt -IXmu -IX11 -lm
```

```
SYS_LIBPATH = -L/usr/lpp/X11/lib/R5/Motif1.2 -L/usr/lpp/X11/Motif1.2/lib
```

```
...
```

Build with external libraries

Imakefile.mk

Link with external libraries

On NT

LOCAL_LDFLAGS = /LIBPATH:"E:\DirectoryWhereTheLibrariesAreStored"

Name of the libraries

SYS_LIBS = LibraryName.lib

Link with include files

LOCAL_CCFLAGS = /I"E:\DirectoryWhereTheIncludeFilesAreStored"

Link with external libraries

On UNIX

LOCAL_LDFLAGS = -L/MachineName/DirectoryWhereTheLibrariesAreStored

Name of the libraries

SYS_LIBS = LibraryName

Link with include files

LOCAL_CCFLAGS = -I/MachineName/DirectoryWhereTheIncludeFilesAreStored

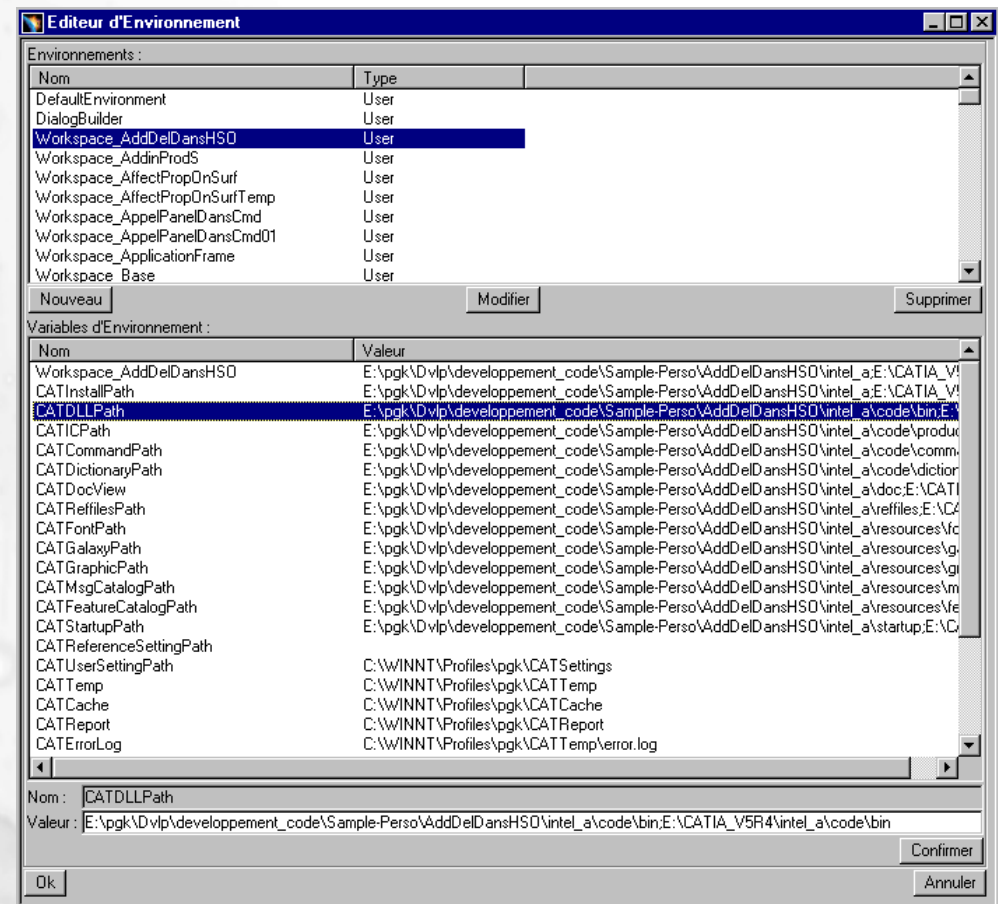
Run with external libraries



Modify environment's variables

➤ On NT use (Start + Programs + CATIA + Tools + CATIA Environment Editor)
E:\CATIAV5_Rx\intel_a\code\bin\CATIAENV.exe

➤ On UNIX modify the csh file
CATENV/CATIA.V5Rx.B0x.csh



About mkmk



Its behavior depends on the current directory:

your workspace directory is the current directory

mkmk -aug → to force all the modules to be rebuilt with the debug option.

mkmk -a → to rebuild only what needs to be rebuilt

A module directory is the current directory:

mkmk -ug → to force the corresponding module to be rebuilt with the debug option.

mkmk → to rebuild only if necessary



To access the mkmk help on line, use mkmk -h.

Mkmk Tips



Use the update (-u) option when:

modifying the dependencies (an include file added or suppressed)

adding or removing a file (.h and .cpp).

modifying the IdentityCard.h and/or the Imakefile.mk



In other cases, do not use the update option. mkmk will reuse some intermediate files generated before.

Objects

ImportedInterfaces

various

ExportedByModuleName Preprocessor Variables



A Windows NT mechanism imposes that shared libraries declare explicitly what they import and export.



To manage this, we define some pre-processor variables.

MyClass.h

```
#include "MyModule.h"
Class ExportedByMyModule MyClass
{
  ...
}
```

MyModule.h

```
#ifdef _WINDOWS_SOURCE
#ifdef __MyModule
#define ExportedByMyModule __declspec(dllexport)
#else
#define ExportedByMyModule __declspec(dllimport)
#endif
#else
#define ExportedByMyModule
#endif
```

Variable defined by
mkmk on Windows NT

Variable defined by mkmk
when building MyModule

Run time tools



mkrtv

copy the application resources (icons, message files, dictionaries, ...) from the Build time directories into the Run time directories.



mkrun

run CATIA V5 or any main executable developed on top of CAA V5

mkrun -c MyProgram

Test Tool: mkodt



mkodt

***Every framework FW should provide its test framework:
FW.tst***

Uses some predefined environment variables

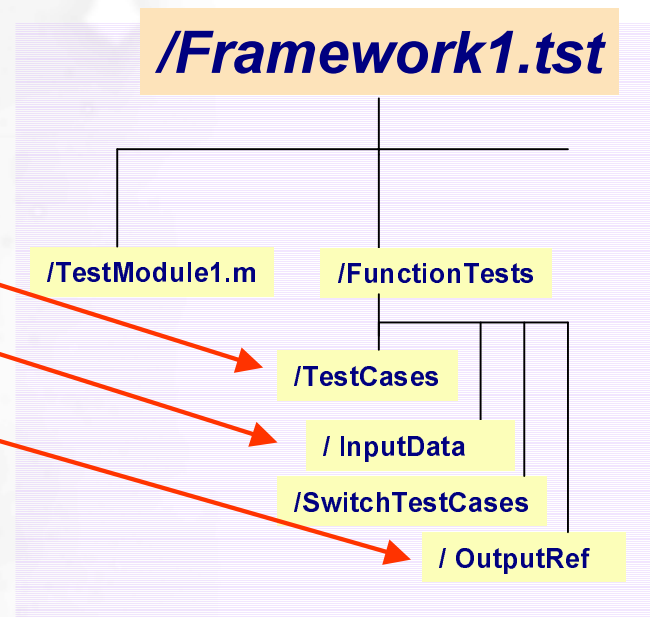
ADL_ODT_OUT,

ADL_ODT_TMP

Contains shells that launches the test programs

Contains any data required by the shells: models, ...

Contains any reference data that can be used by the shells to check what is produced by a test program









C++ Source Checker: mkCheckSource



mkCheckSource can detect corrupting errors before dynamic tests. This insures a better way of programming, reducing development and debug time.



Checked Rules :

-  **AddRef / Release**
-  **Callbacks**
-  **Pointer life cycle**
-  **Exceptions**
-  **C++**
-  **Life cycle of some particular objects (Dialog ...)**

Source Code Manager (1/3)



SCM is a Software Configuration Management System designed for supporting CAA V5 developments.

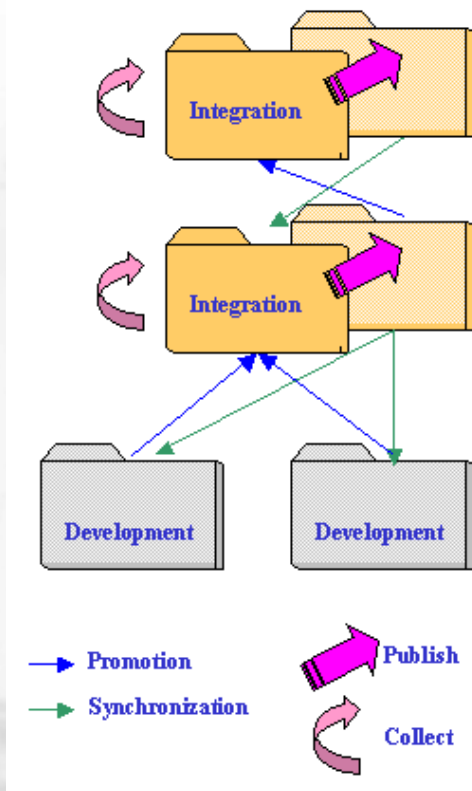


Throughout workspaces' features SCM provides a way for a company to organize and to control software developments between development departments.



Users work on files on their personal file systems and interact with each others by commands involving the whole workspace.

Source Code Manager (2/3)

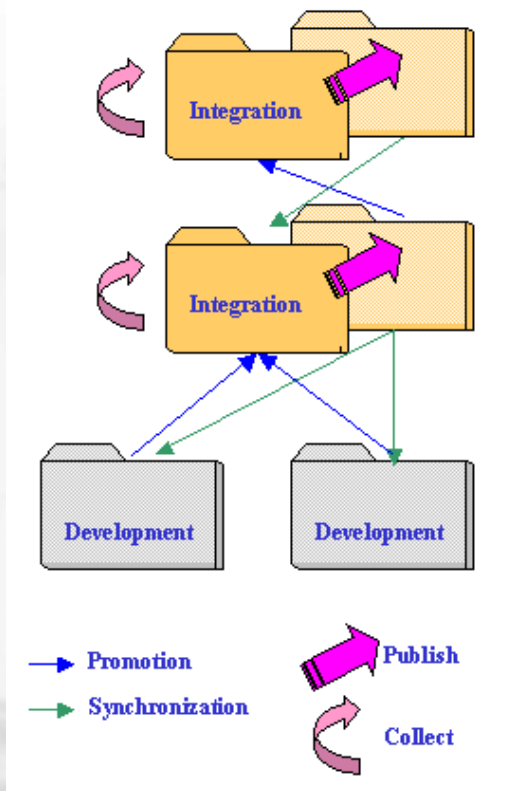


The process is based upon four main operations:

▣ *the promotion: it is the delivery of developments from a workspace to its parent workspace*

▣ *the collect: it is the action of getting in a workspace what has been delivered by child workspaces*

Source Code Manager (3/3)



the publication: it consists in making available to other workspaces what was in the private part of a project workspace. Available changes are in the public part and child workspaces can get them with the synchronization.

the synchronization: it is the action of getting in a workspace the last state of the component managed in this workspace from the public part of the parent workspace.

Microsoft Developer Studio CAA V5 Add-Ins



All our specific tools have been integrated in Microsoft Developer Studio V6

Must be installed using the Unicode String option.

```
Copyright DASSAULT SYSTEMES 2000
// EAEHrCombinedCurveEdit.cpp
// Provide implementation to interface
// CATIEdit
//
// Local Framework
#include "CAAEHrCombinedCurveEdit.h"
#include "CAAEHrCombCrvPanelStCmd.h" // needed to return the Combined Curve edition command
// CAAMechanicalModeler edu Framework
#include "CAAEHrCombinedCurve.h" // needed to build the command from the Combined Curve
CATIImplementationClass ( CAAEHrCombinedCurveEdit :
    DataExtension
    CATBaseUnknown
    CombinedCurve
);

// CAAEHrCombinedCurveEdit : constructor
CAAEHrCombinedCurveEdit::CAAEHrCombinedCurveEdit()
{
    CATBaseUnknown()
}

// CAAEHrCombinedCurveEdit : destructor

// end step: DataUpdate at 06/20/2011-18:53:58
// Command done, return code = 0
Tool returned code: 0
```

CAA V5 wizards in Ms Developer Studio



Wizards to generate code corresponding to generic tasks:

New CAA V5 Workspace

New Framework

New Module

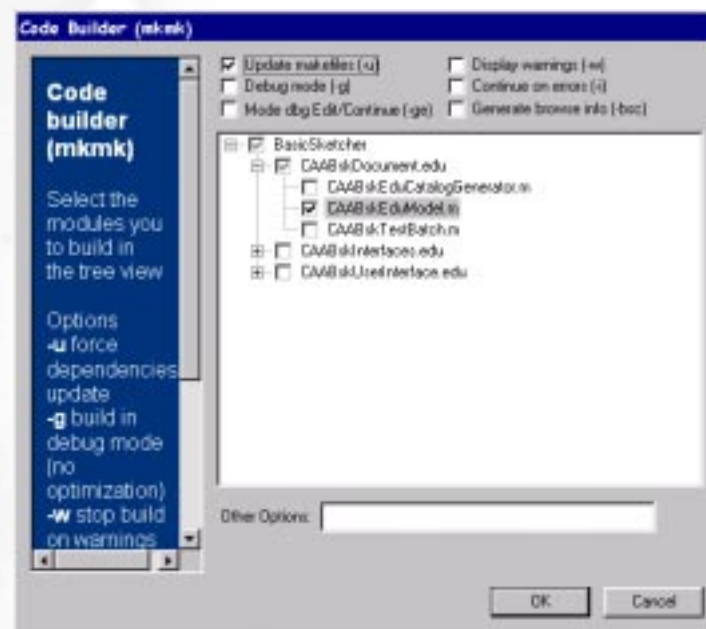
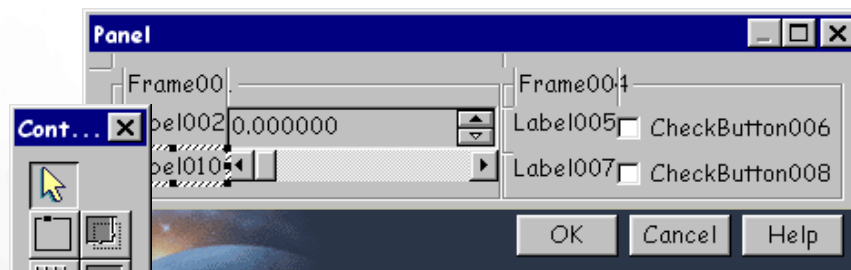
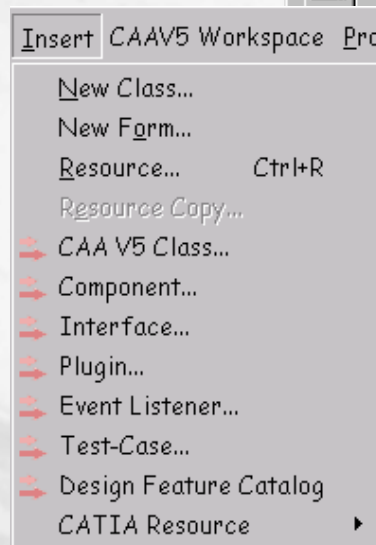
New Command

New Panel

New Interface

New implementation

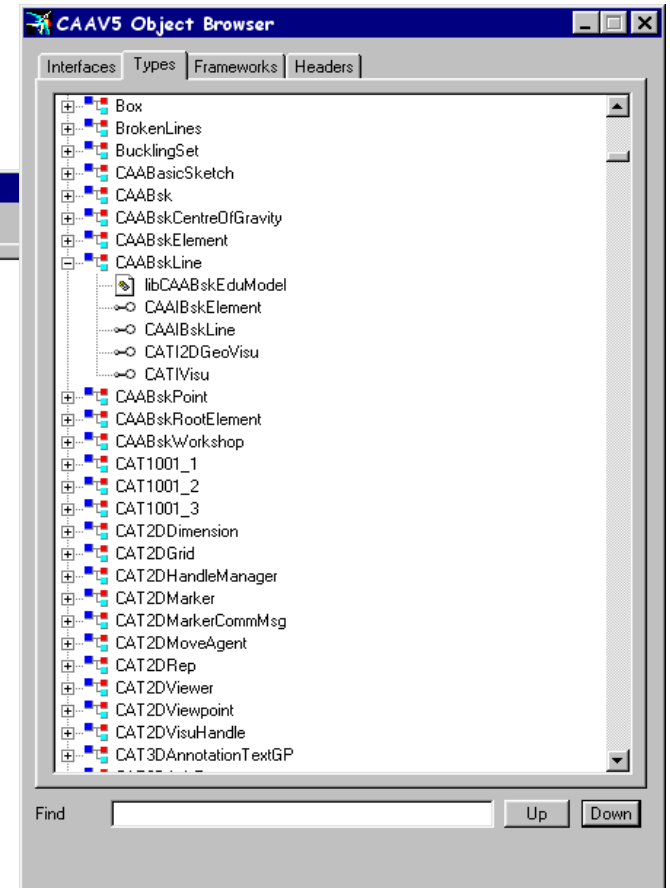
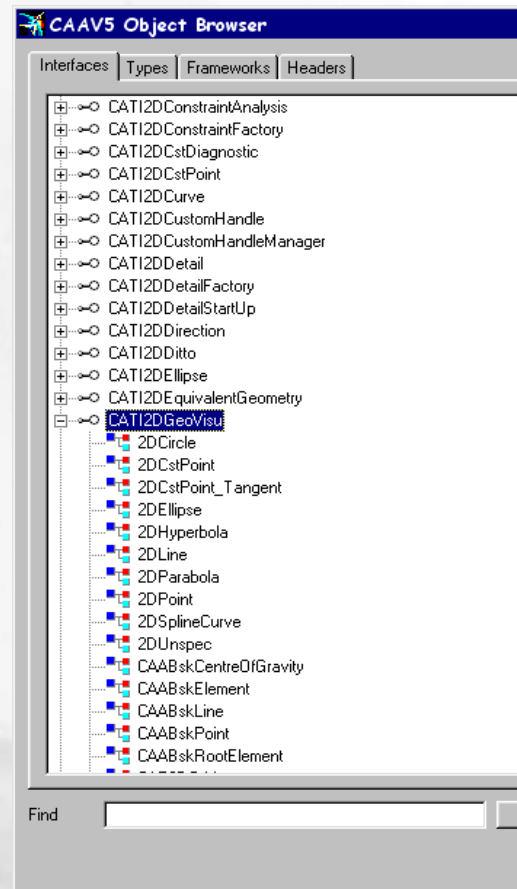
...



CAA V5 Object Browser



To know which component implements which interface



Mapping between commands and MsDev Add-Ins

mkGetPreq / mkCopyPreq

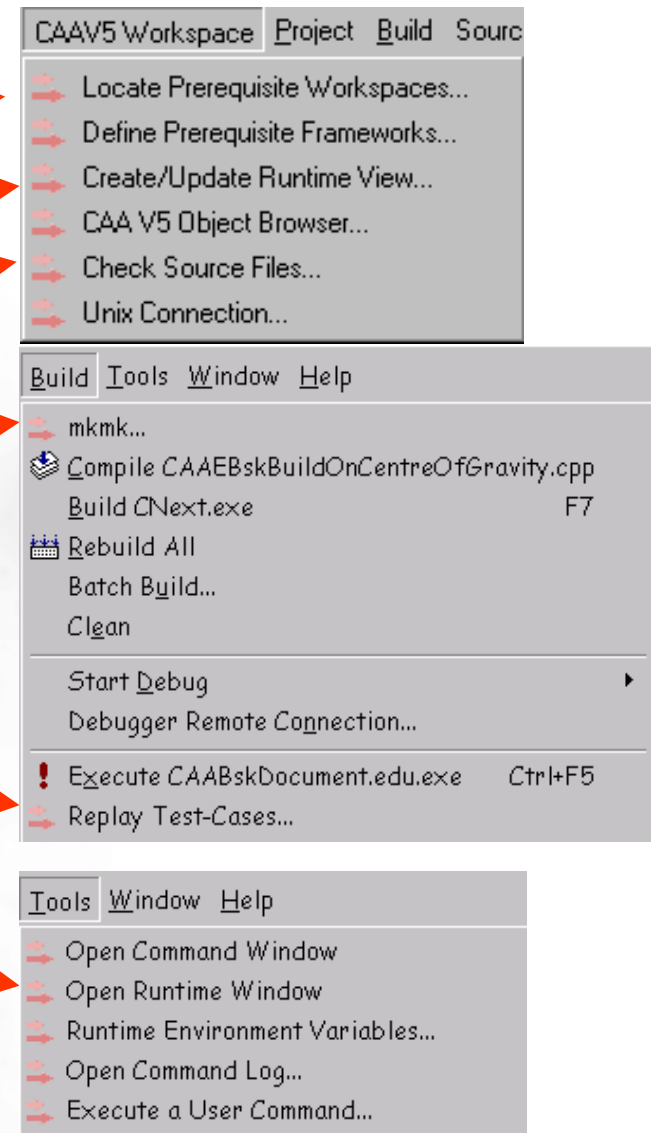
mkrtv

mkCheckSource

mkmk

mkodt

mkrun



Mapping between commands and MsDev Add-Ins

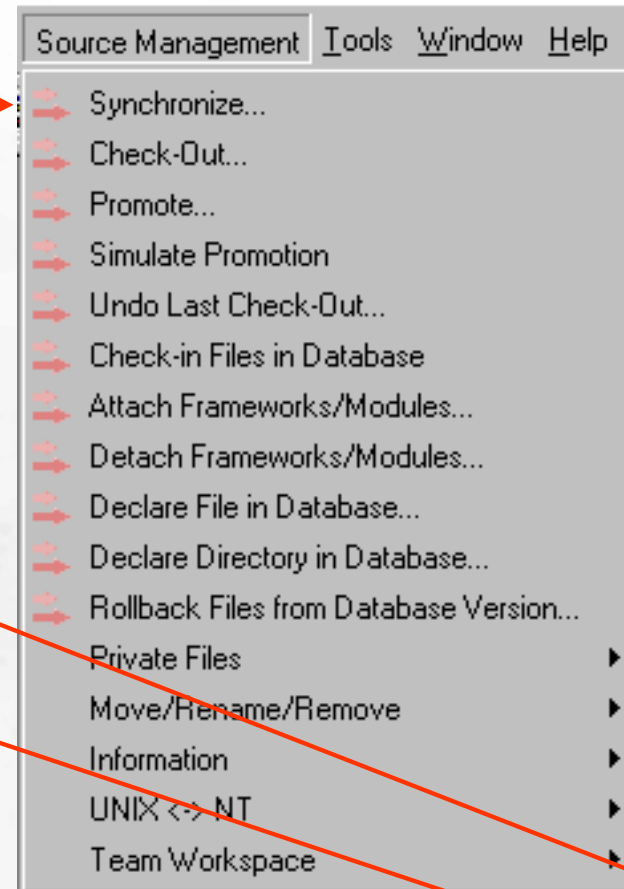
Software Configuration Management

Synchronize

Promote

Collect

Publish

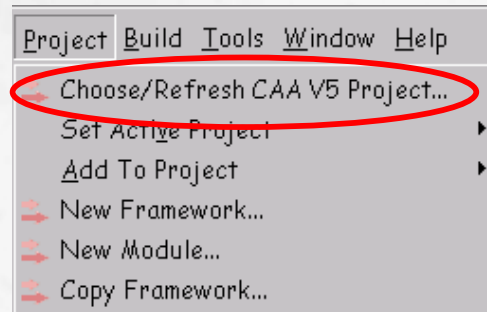


Collect...
Publish...

MSDev Add-Ins: Hints and Tips (1/3)



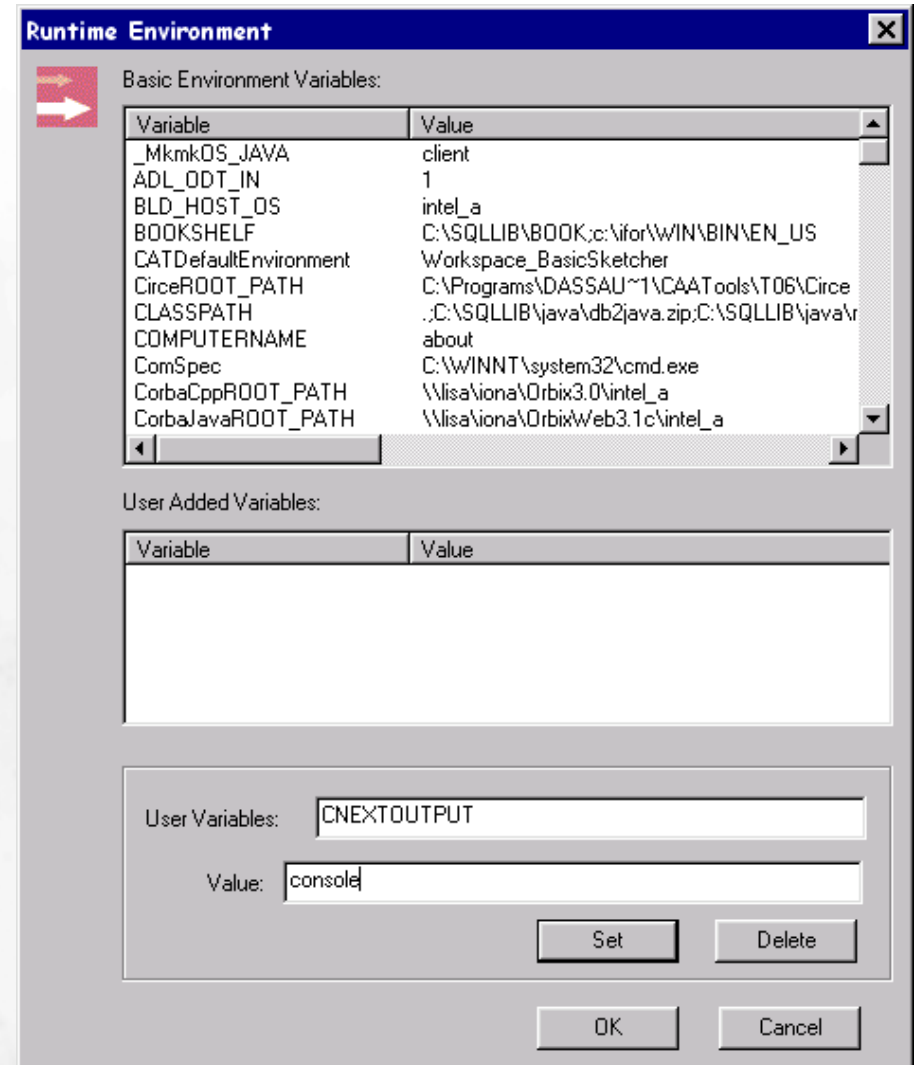
***To be able to see any modification done directly in NT Explorer (copied, moved or deleted files) in MSDev,
Use command **Project+Choose/Refresh CAA V5 Project...*****



MSDev Add-Ins: Hints and Tips (2/3)



*To be able to see your trace statements the environment variable **CNEXTOUTPUT** has to be set to console in Tools+Runtime Environment Variables*



MSDev Add-Ins: Hints and Tips (3/3)



To rebuild a module and if you don't need the update option, use the keyboard shortcut F7.



To export a workspace (just the source code) get rid of all the intermediate files generated by mkmk: go to Tools + Open Command Window and key `mkRemoveDo -a`



*[Ctrl-Q] to swap between .h and .cpp files
[Ctrl-T] to open the .h file corresponding to the keyword under the cursor
[Ctrl-F1] for API documentation*

Enable porting on UNIX from Visual C++

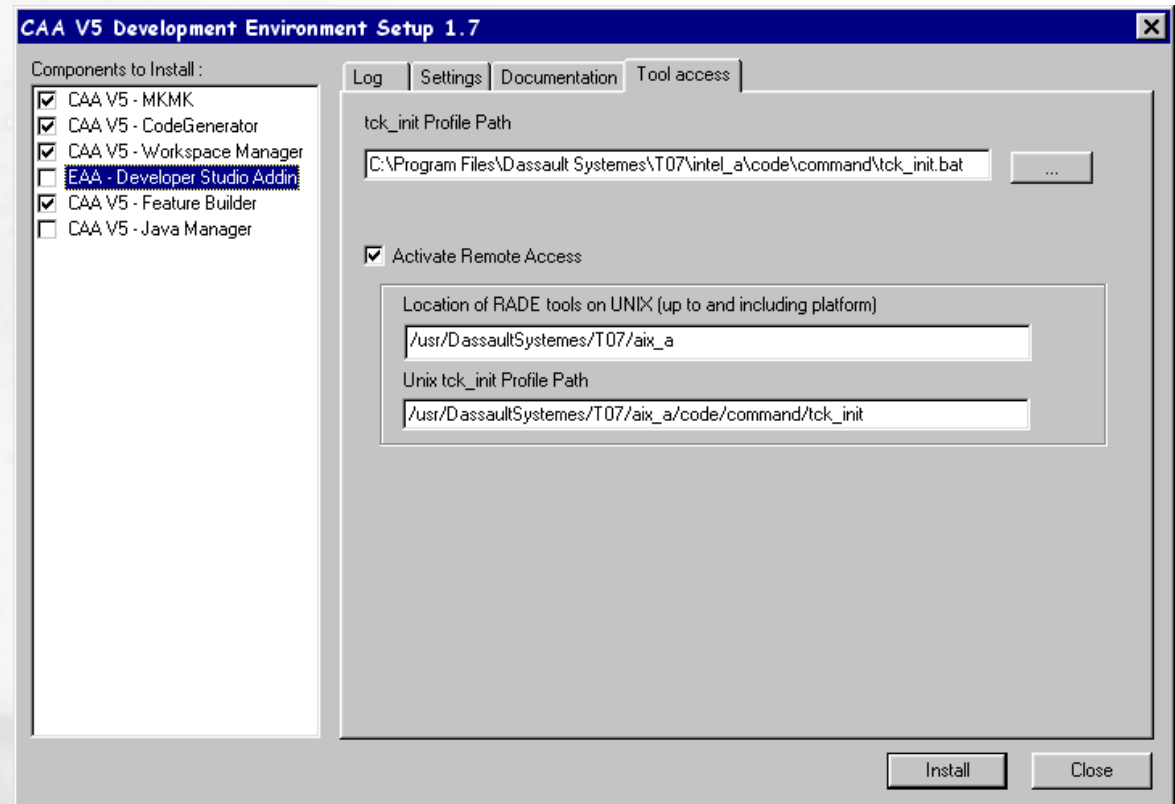


If you need to port your applications on UNIX, you should run again the CATVBTSetup executable:

C:\Program Files\Dassault Systemes\T07\intel_a\code\bin



In the Tool access tab page, activate the remote access and inform where the CAA V5 Tools are installed on UNIX



Activate the Porting on UNIX



When opening a workspace, you can ask for building on UNIX by informing Visual C++ on which UNIX machine, with which user and in which directory the operations will be performed.



Later on whenever a file is generated on NT, it is copied meanwhile on UNIX

Open CAA V5 Workspace

Open with: Mkmk

Workspace Directory: E:\Training\BasicSketcher

Tool level: V5R6_T06

☒ Build On UNIX

UNIX Host: abigdsy UNIX Login: ctd

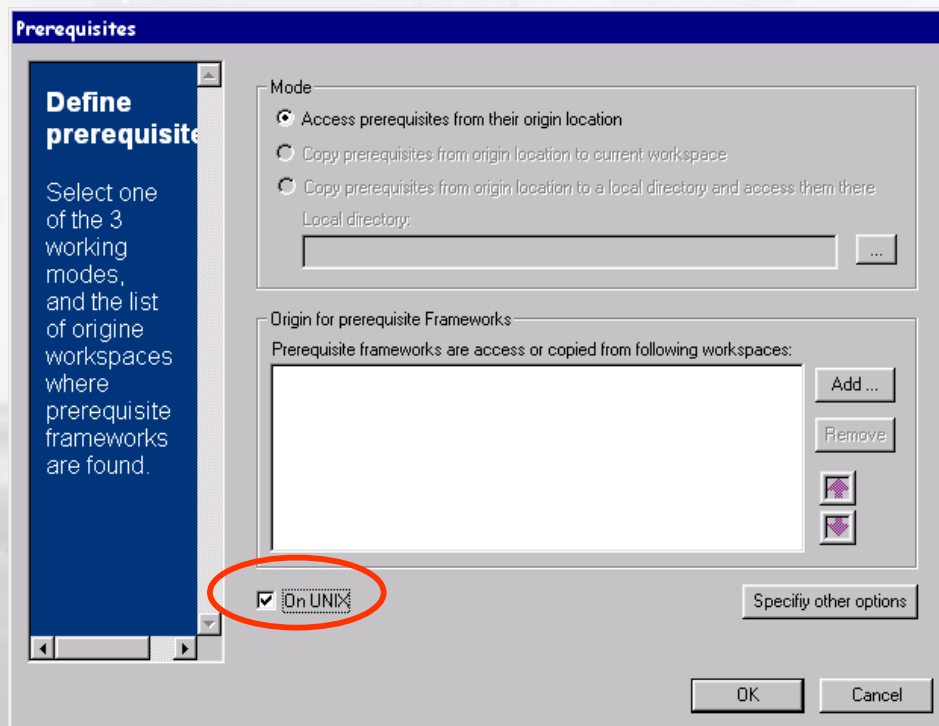
Directory for build: /u/users/ctd/MyWorkspaces

OK Cancel

Porting on UNIX



*From Visual C++, then
you can define the pre-requisite workspaces
you can build
you can update the run time view*



Other Tools used in the CAA V5 context



Rational Purify

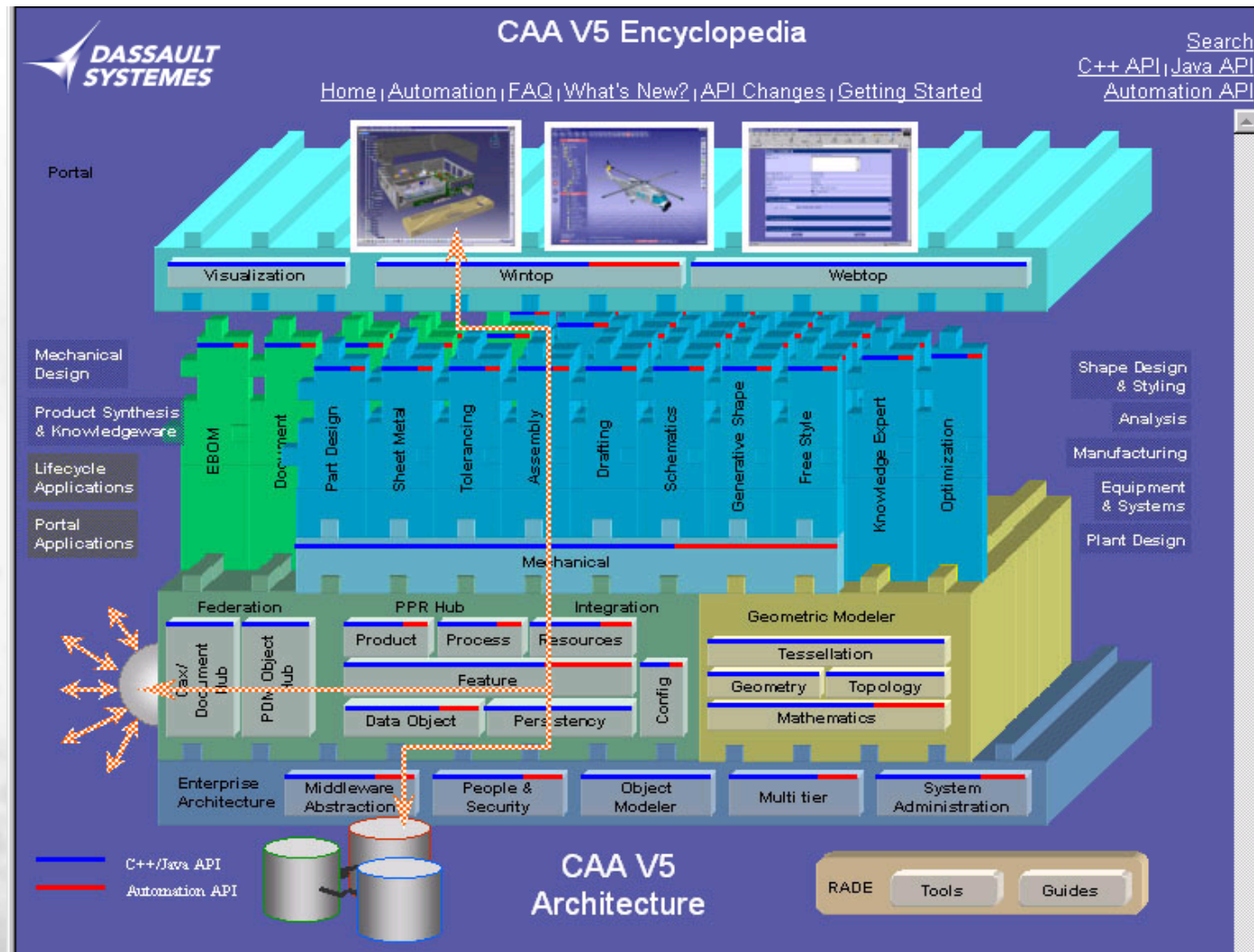
A tool to detect any memory leak and to be used with mkodt



Rational Pure Coverage

A tool to check the percentage of the code really tested and to be used with mkodt

CAA V5 Encyclopedia Home Page



CAA V5 C++ Objects Documentation



Interface GeometricObjects.CATNurbsCurve

```
System.IUnknown
|
+---System.IDispatch
|
+---System.CATBaseUnknown
|
+---GeometricObjects.CATIGMObject
|
+---GeometricObjects.CATGeometry
|
+---GeometricObjects.CATCurve
|
+---CATNurbsCurve
```

Usage: an implementation of this interface is supplied and you must use it as is. You should not reimplement it.

interface CATNurbsCurve

Interface representing a Nurbs curve.

A CATNurbsCurve is created by the CreateNurbsCurve method of the CATGeoFactory interface and deleted with the Remove method. It is defined with:

CATKnotVector	KnotVector	The knot vector for the polynomial basis definition
CATMatSetOfPoints	Vertices	The set of control points
CATBoolean	IsRational	TRUE if the nurbs is rational, FALSE otherwise
double[]	Weights	The array of weights if IsRational

See also:

[CATKnotVector](#)

CAA V5 Programmer's Guide



All documentations about a domain

Technical Articles

Use Cases

Quick references

CAA V5 Programmer's Guide



Technical articles

In depth paper

Less than 10 pages

Hyper linked

DASSAULT SYSTEMES

Application Frame Overview
The basics of interactivity

Application Frame Programming

Version: 1 [Jan 2000] [Document history](#)

Abstract

This article explains which paradigms CNext uses to show objects and let end users play with them.

- [The Anatomy of a Typical CNext Application Window](#)
- [The Application Window](#)
- [The Document Window](#)
- [Workshops and Workbenches](#)
- [Interactive Commands and Command Headers](#)
- [How Commands Are Presented to the End User](#)
- [Transitions between Workbenches](#)
- [Objects Providing the Interactive Behavior](#)
- [In Short.](#)

The Anatomy of a Typical CNext Application Window

The CNext application window is the host for all CNext documents. CATIA, as an MDI (Multiple Document Interface) application, can display several document windows at the same time as child windows of its application window, one document window, and thus one document, being active at the same time. Have a look at the screen shot below.

CATIA V5.2 The standard menubar The application window

Start File Edit View Insert Tools Window Help

CAA V5 Programmer's Guide



Use Cases

CAA V5 Code in Action

Step by Step

Each step detailed and commented

Delivered with fully operational source code

Made to be copied/pasted into customer code

2. Create the CAAfrGeometryWks.cpp file. The file skeleton is shown below. The implementation of each method is described in separate sections.

```
#include "CAAAfrGeometryWks.h"
#include "CATICAAfrGeometryWksConfiguration.h"
#include "CAAIAfrGeometryWksAddin.h"

#include "CATCommandHeader.h" // See Creating the Command Headers
MacDeclareHeader(CAAfrGeometryWksHeader);
#include "CAAAfrDumpCommandHeader.h"

#include <CATCreateWorkshop.h>

CATImplementClass(CAAfrGeometryWks, Implementation, CATBaseUnknown, CATnull);
#include <TIE_CATIWorkshop.h>
TIE_CATIWorkshop(CAAfrGeometryWks);

CAAAfrGeometryWks::CAAAfrGeometryWks() {}
CAAAfrGeometryWks::~CAAAfrGeometryWks() {}

void CAAfrGeometryWks::CreateCommands()
```

CAA V5 Programming Rules



Programming Rules

Naming convention

C++ coding rules

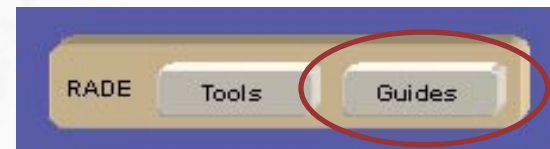
Java coding rules

Architecture rules

User Interface look and feel



Available in the encyclopedia



CAA V5 Naming Convention



Naming conventions

To avoid name collisions

To make things clearer for its developers



Names are constituted by English names, each one starting with an uppercase.



Three letters alias for product name.

CAT / VPM / ENOV / DNB reserved for DS product lines



Three letters alias for each framework.

CATMoldDesignFeature (framework)

CATMldComponent.m (module)

CATMldEjectorImpl.cpp/.h (class)

CAA V5 C++ Programming Rules



Prefer class forward declaration to #include



Avoid multiple inheritance



Use variable naming convention

Argument prefix: i for input, o for output

Variable prefix:

<i>p</i>	<i>pointer</i>
<i>pp</i>	<i>pointer on pointer</i>
<i>sp</i>	<i>smart pointer</i>
<i>pi</i>	<i>pointer on interface</i>
<i>a</i>	<i>array</i>



Do not use friend class

To Sum Up ...

In this lesson you have seen...

- *The CAA V5 directory tree structure*
- *The specific tools developed on top of the standard compilers to speed up the development*
- *The integration of these specific tools in Microsoft Developer Studio as Addins.*
- *How we develop on Windows NT and how we port on UNIX*

CAA V5 ObjectModeler

In this lesson you will learn the CAA V5 OO infrastructure.

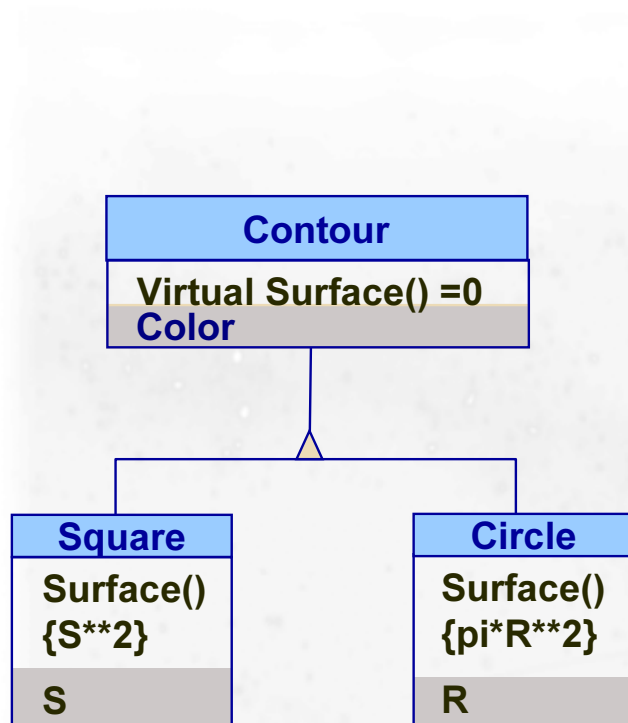
- **Why a New Object Modeler?**
- **The Interface / Implementation Pattern**
- **Between Interfaces and implementations**
- **Object Life Cycle Management**
- **Extension Mechanism**
- **Late Typing**

Why a new Object Modeler?

You will learn why the C++ object model alone does not suit our needs.

- ▣ C++ as a Starting Point
- ▣ What's wrong with C++?

C++ as a Starting Point



Encapsulation

Hiding implementation details

Changing private part without client impact



Inheritance

Reusing implementation

Extending types



Polymorphism

Genericity on client side

Specialization of provider side

What's wrong with C++ ?



Encapsulation could be better

*Compilation link between object and its client
even if only a private member changes*



What's wrong with C++ ?



Not enough run time flexibility

Cannot instanciate classes by name

```
void* myObj = new(iargv[1])
```



Needed for

Adding new types to the system...

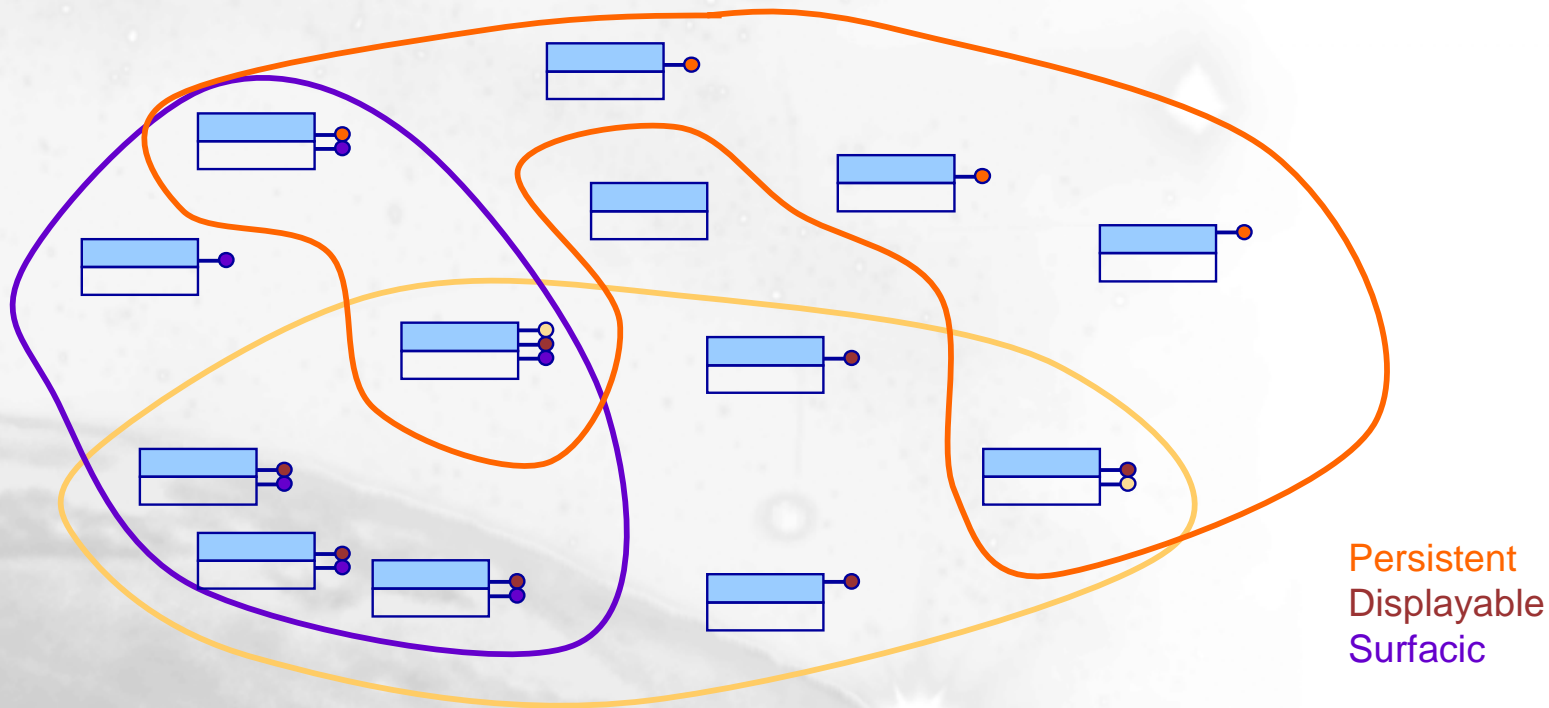
... and letting « old » system instanciate them

What's wrong with C++ ?



No support for «schizophrenic» objects

In large systems, objects have more than one type. These types are needed or useless according to the context. Multiple inheritance is not a solution



What's wrong with C++ ?

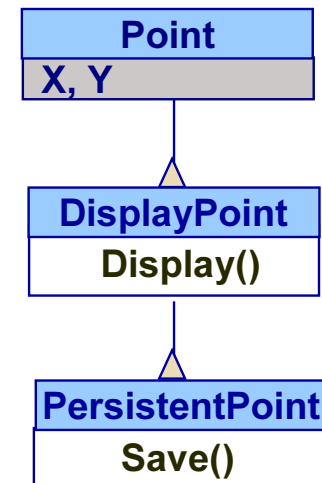
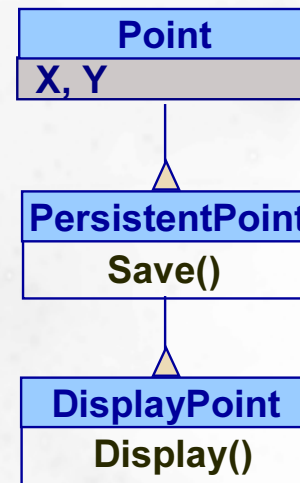
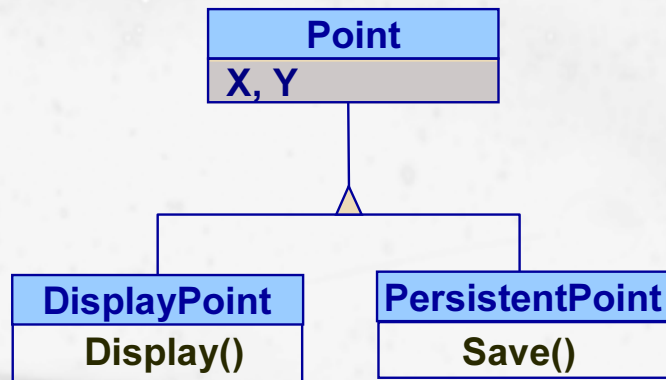


Extension only through inheritance

How to organize inheritance trees?

What if extensions added later?

What about code modularity?



The Interface / Implementation Pattern

You will learn the Interface handling, shielding any client application from implementation details

- ▣ Interface / Implementation Pattern

- ▣ Interface

- ▣ Definition

- ▣ Using and retrieving Interfaces

- ▣ Coding an Interface

- ▣ Implementation

- ▣ Definition

- ▣ Coding an Implementation

- ▣ Implementation Inheritance

- ▣ Factory

- ▣ Definition

The Interface / Implementation Pattern



Client and Implementation separation is achieved by the mean of interfaces.



Interface shields the client object from the implementation details.



Alternative notation :



Interface



An interface is an abstract object with a set of pure virtual methods that defines a behavior that objects are subject to support.



An interface does not know which objects implement it.



An interface sets a contract between a client and provider's code

Function prototypes

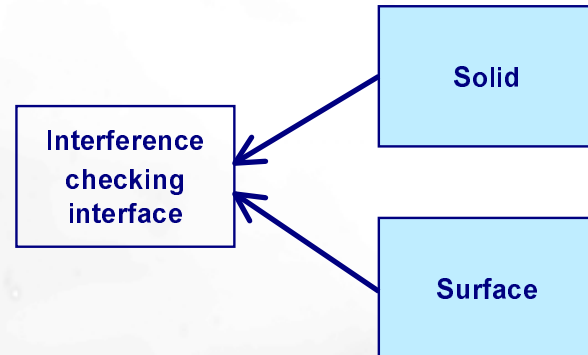
Design intent



All interfaces inherit from IUnknown / CATBaseUnknown

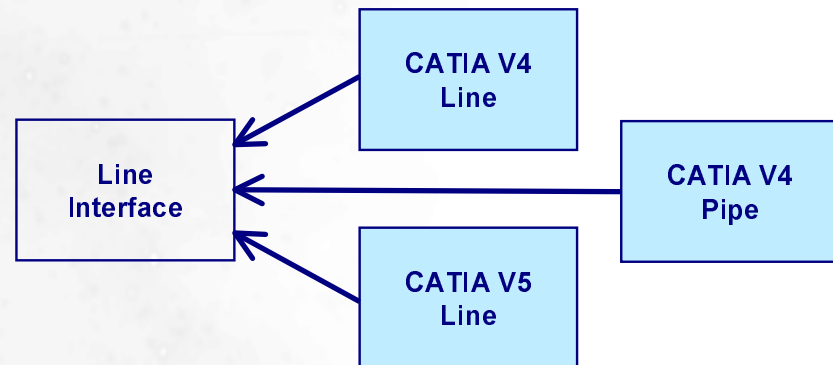
Behavior or Abstract Data Type

➔ ***A common behavior shared by many objects***



□ Interfaces
□ Implementations

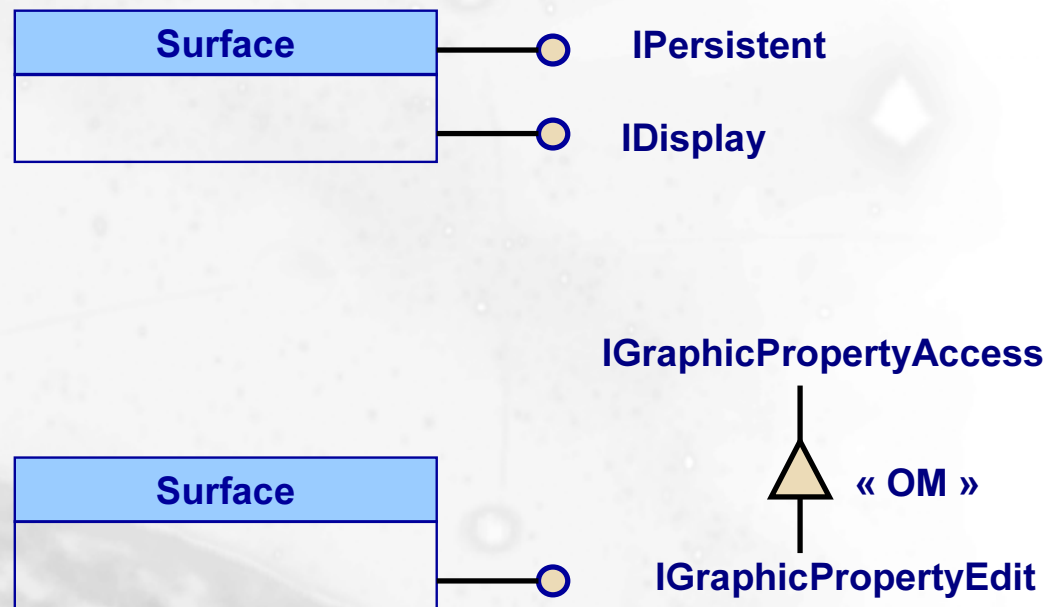
➔ ***An abstract data type implemented by many objects***



Interface Inheritance



Interfaces can inherit from each other



Using Interfaces

An interface

CATITransform

```
virtual Translate()=0  
virtual Rotate()=0  
virtual Move()=0
```

How an application uses it

```
#include "CATITransform.h"  
[...]  
CATITransform * transformInterface;  
// [...Here we get the instance of the interface...]  
if (transformInterface)  
{  
    CATVector3f transVector(10., 10., 10.);  
    transformInterface->Translate (transVector);  
    cout << "Element translated" << endl;  
}
```


Translate service is asked to the interface, not to the implementation.

- Application uses interfaces **instead of** implementation objects.
- The application remains **isolated** from implementation.

... but how to get an interface ?

Getting an Interface from an Implementation

```
[...]  
// Query if point implementation can provide a CATITransform interface  
CATITransform * transformInterface = NULL;  
HRESULT rc = pointImpl->QueryInterface(IID_CATITransform,  
                                         (void*)& transformInterface);  
// Use returned interface to manipulate point  
if SUCCEEDED (rc)  
{  
    transformInterface ->Translate(...);  
    cout << "Point translated" << endl;  
}  
[...]
```



QueryInterface returns as its second argument an instance of *CATITransform* if *pointImpl* implementation instance adheres to it, *NULL* otherwise.

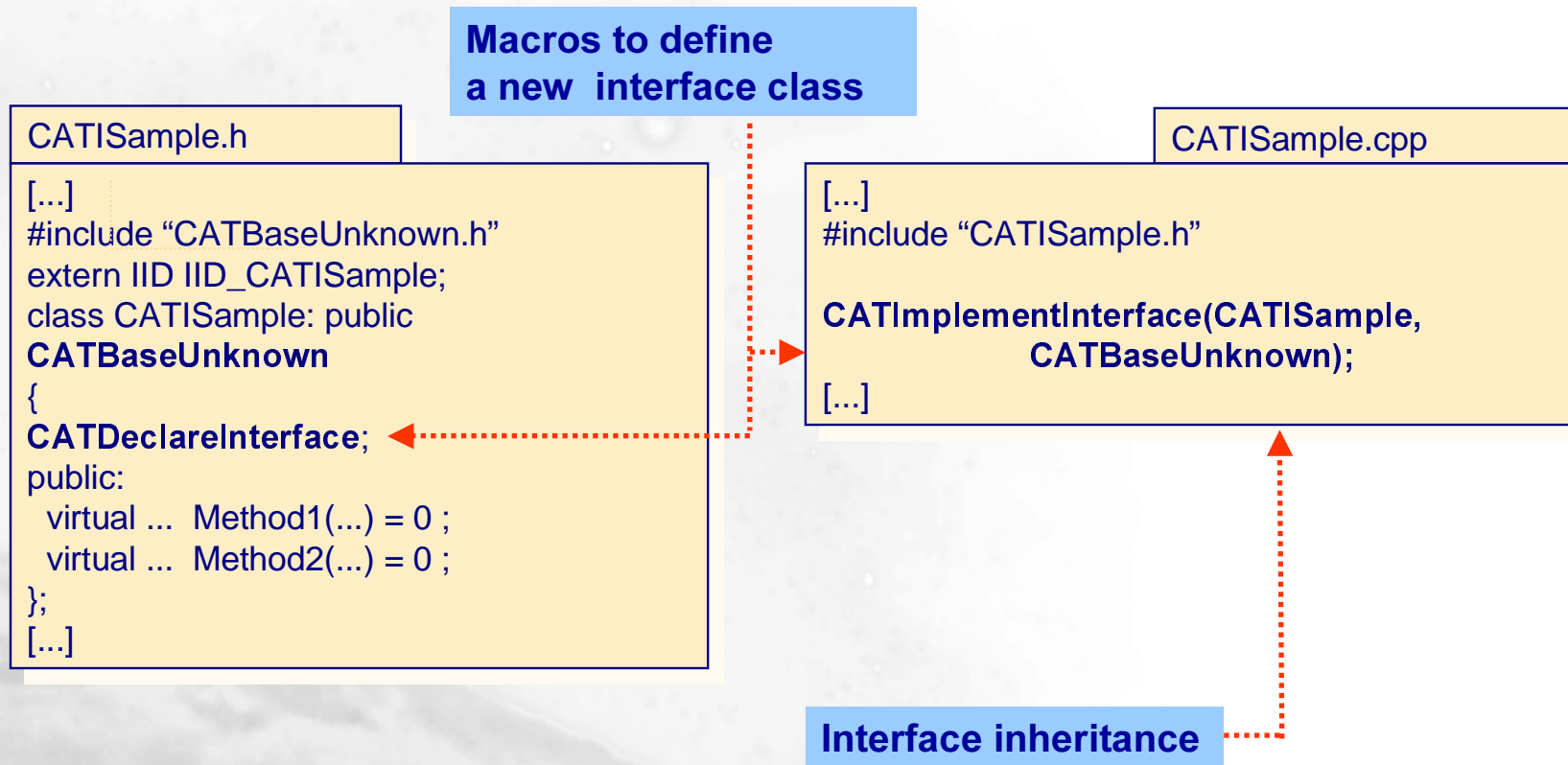
However, clients should not deal with implementations to avoid coupling. This capability is offered for special purposes like factory programming.

Getting an Interface from another Interface

```
[...]  
// Query if CATIVisu interface is supported in addition to CATITransform  
HRESULT rc = transformInterface->QueryInterface(IID_CATIVisu,  
        (void**)& visulInterface);  
// Use returned interface to manipulate point  
if SUCCEEDED (rc)  
{  
    visulInterface->Draw(...);  
    cout << "Point displayed" << endl;  
}  
[...]
```

QueryInterface also works with an existing interface instance as input, i.e. an interface instance can be queried against adherence to another interface just as the implementation object it stands for.

Interface at work



Interface at work : the GUID



The Global Unique Identifier (GUID) of an Interface is stored from V5R6 in a specific module :

Module's default name generated by CAA wizard :

MyFrameworkInterfacesUUID.m

Name of the file :

CATISample.cpp

CATISample.cpp

```
#include "IUnknown.h"
extern "C" const IID IID_CATISample = { 0x32bbc9aa, 0x0254, 0x11d5, { 0x85, 0x08, 0x00, 0x02,
0xb3, 0x06, 0x11, 0x71} };
```

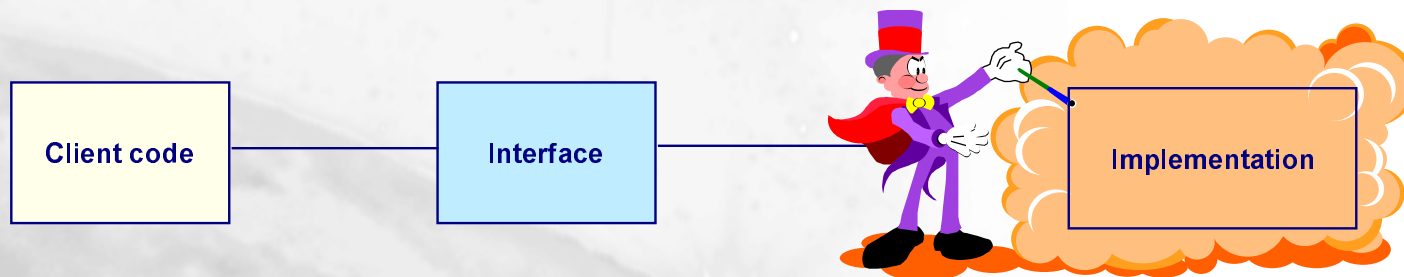
To compute a GUID
on UNIX: `uuid_gen -c`
on NT: `uuidgen -s`

The MyFrameworkInterfacesUUID module has to be included in the Imakefile.mk files in other Frameworks for accessing CATISample interface

Implementation

An implementation is an object that defines a specific way to fulfill the contract set by an interface.

- ➔ An implementation has to explicitly state which interface(s) it adheres to.
- ➔ An implementation provides code for all the abstract methods defined in the interfaces it adheres to.
- ➔ An implementation may adhere to many interfaces.
- ➔ The client application deals with the implementation only through the interface.



Implementation at work

Macros for an implementation class

ImplSample.h

```
[...]
#include "CATBaseUnknown.h"

class ImplSample: public
CATBaseUnknown
{
  CATDeclareClass;

  public:
    ImplSample() ;
    ~ImplSample() ;

    // CATISample adhesion
    ... Method1(...) ;
    ... Method2(...) ;

    // Other methods
    [...]
};
```

ImplSample.cpp

```
[...]
#include "ImplSample.h"

CATImplementClass ( ImplSample,
Implementation,
CATBaseUnknown,
CATNull )

[...]
// CATISample adhesion
[...]
... ImplSample::Method1() { ... }
... ImplSample::Method2() { ... }

// Other methods
[...]
```

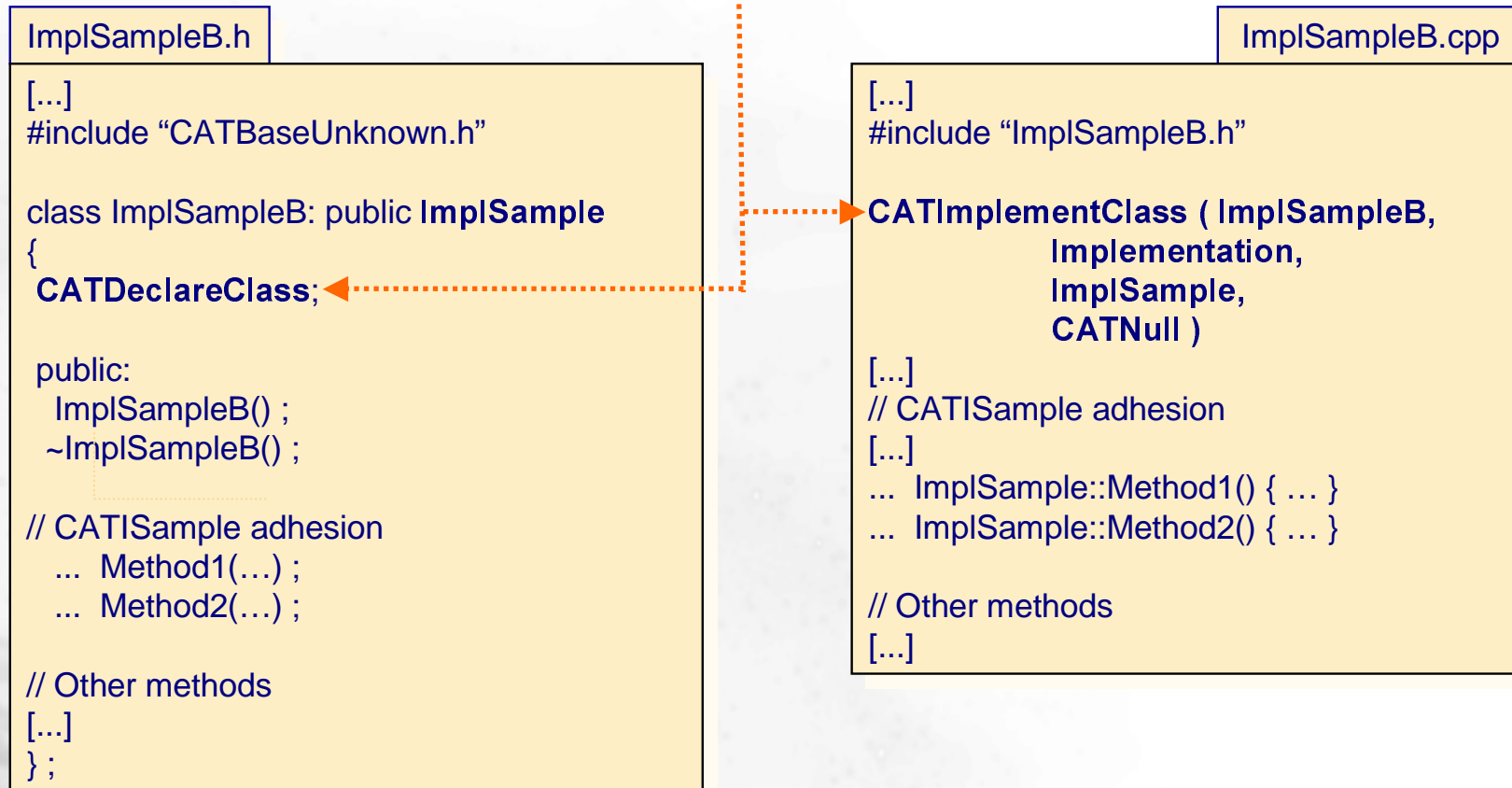
ImplSample

0

CATISample

Implementation at work

Macros for an implementation class with inheritance

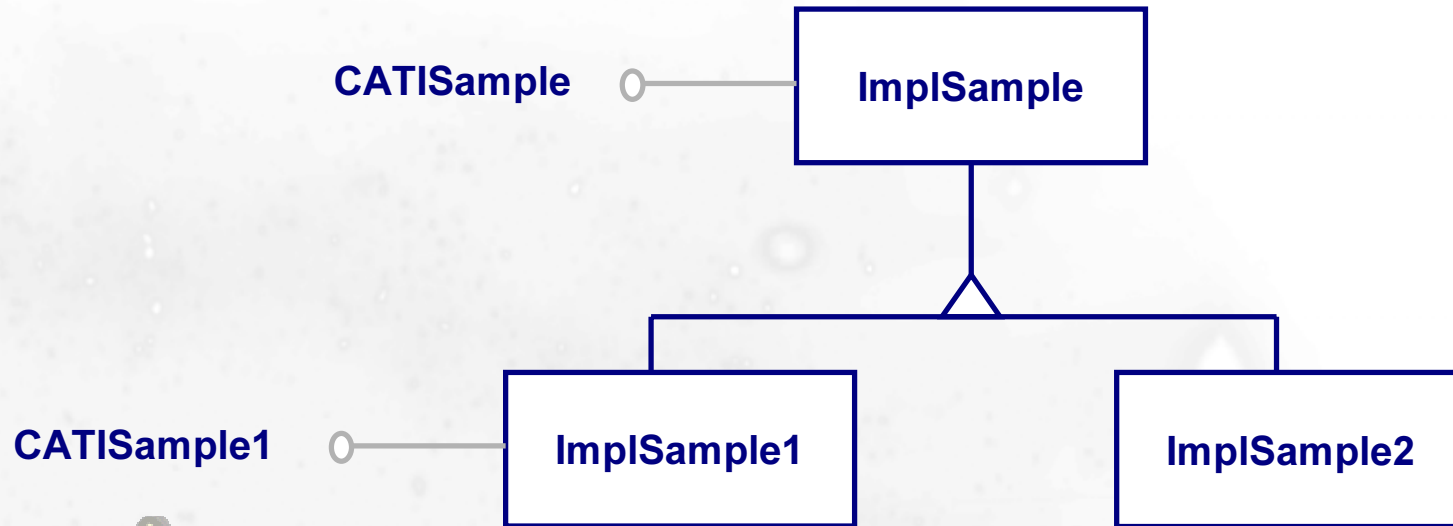


The CATImplementClass Macro

```
CATImplementClass ( <this ClassName>,  
                   <its OM type>,  
                   <its OM inheritance | CATBaseUnknown>,  
                   <what it extends | CATNull  )
```

- 
- 3 main types:
- Implementation
 - CodeExtension
 - DataExtension

Implementation Inheritance



ImplSample1 adheres to CATISample1 and to CATISample by inheritance from ImplSample whereas ImplSample2 adheres only to CATISample.

Factory

A factory is a special object that contains methods **dedicated to object creation**.



The factory creates an implementation object and returns an interface object on it.



The factory minimizes the coupling since the client application doesn't manipulate any implementation instance.



The object creation is centralized which enables a better object management.

Factory at work within a client application

Interfaces on factory are usually retrieved through a QueryInterface on a high level object (documents, containers...).

```
[...]  
  
// Retrieve the Factory  
CATICatalogFactory *myFactory = ...;  
CATICatalog *myCatalog = myFactory->CreateCatalog("CatalogA");  
  
[...]
```

The **CreateCatalog** method creates an implementation object and return an interface on it to the client application.

Between Interface and implementation

You will learn how the link between interface and implementation is done through TIE objects.

- ▢ TIE Definition
- ▢ Two types of TIE
 - ▢ Standard TIE
 - ▢ Chained TIE
- ▢ TIE Generation
- ▢ TIE at work

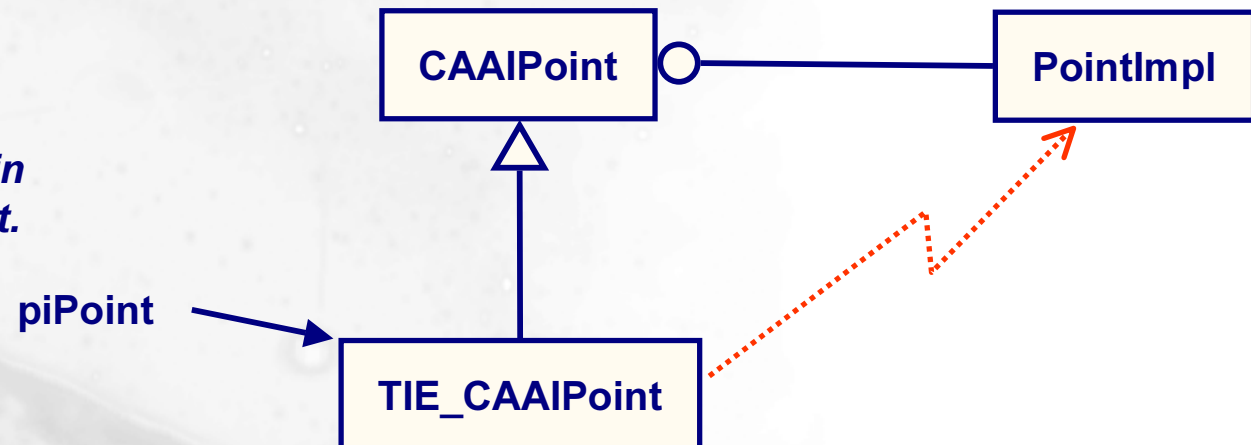
TIE Definition

A TIE is the object that links the interface and the implementation.



Every request to an interface method is redirected to the corresponding implementation method through the TIE.

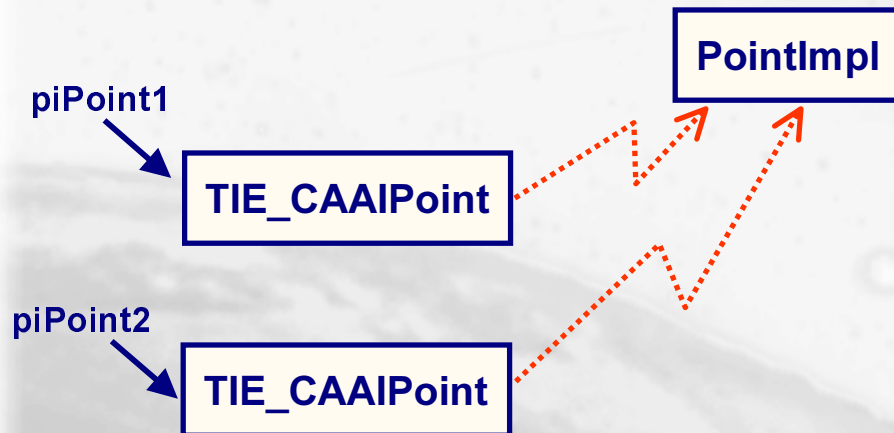
A pointer on an interface is in fact a pointer on a TIE object.



Standard and Chained TIE

Standard TIE

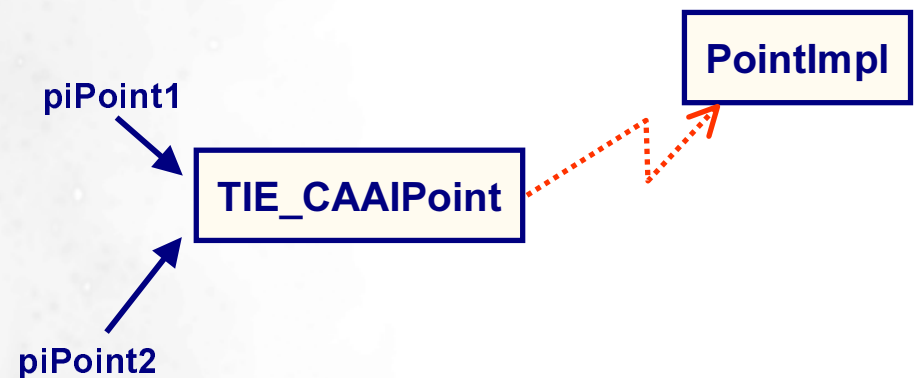
When a pointer to an interface is required, a new TIE is retrieved even if there is already one for the underlying implementation object.



Chained TIE

When a pointer to an interface is required, an existing one is retrieved if possible.

- Better memory usage
- Slower query interface



TIE Generation



If the interface is defined in C++, the TIE header file will be generated if you create a file `TIE_CATlxxx.tsrc` that just includes the interface header file `CATlxxx.h`.

- `TIE_CATlxxx.tsrc` is generated by a wizard under Visual Studio***

`<FWInterfaces>/FWItfCPP.m/CATlPoint.tsrc`

```
//Code Generated by the CAA Wizard
//This source file insures the regeneration of the tie TIE_CAAIPoint.h
#include "CAAIPoint.h"
```

mkmk

`<FWInterfaces>/ProtectedGenerated/intel_a/TIE_CATlPoint.h`

...

TIE at work

TIE at work



The TIE_xxx macro is used for a standard TIE.



The TIEchain_xxx macro is used for a chained TIE.

ImplSample.cpp

```
[...]
#include "ImplSample.h"

CATImplementClass ( ImplSample,
                    Implementation,
                    CATBaseUnknown,
                    CATNull )

[...]
// CATISample adhesion
#include "TIE_CATISample.h"
TIE_CATISample(ImplSample);

... ImplSample::Method1(...) { ... }
... ImplSample::Method2(...) { ... }

// Other methods
[...]
```

Statement defining *ImplSample* adhesion to *CATISample* interface using a standard TIE.

Object Life Cycle Management

You will learn how to manage the life cycle of interfaces and implementations through the Reference counting and smart pointer techniques.

- ▣ **Managing the Object Life Cycle**

- ▣ **Reference Counting**

- ▣ **Theory**

- ▣ **Examples**

- ▣ **Golden Rules**

- ▣ **Smart Pointers**

- ▣ **Theory**

- ▣ **Recommendation**

- ▣ **Do and do not**

Managing the Object Life Cycle

CNext Object Modeler rules

- ❑ Implementation should be manipulated by applications only through interfaces.
- ❑ An implementation can only be deleted when its last pointing interface is deleted.
- ❑ The CNext ObjectModeler deletes automatically implementation.

Who should delete objects and how ?

Managing the Interface Life Cycle



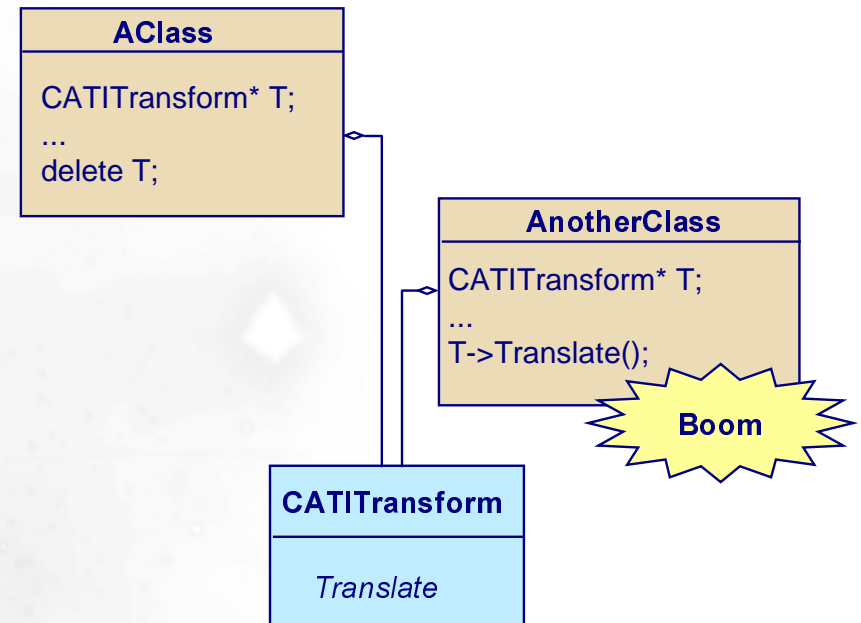
Many client objects can hold a reference on the same interface at the same time.



One client object usually holds more than one reference to an implementation at a time



If one of these clients decides to delete the interface through its reference, what happens to others?



ObjectModeler offers two mechanisms that secure the existence of an interface used by objects.

Reference Counting



Interface offers special methods for locking itself from deletion by others



A client of an interface increments the reference count on the interface while it needs it.



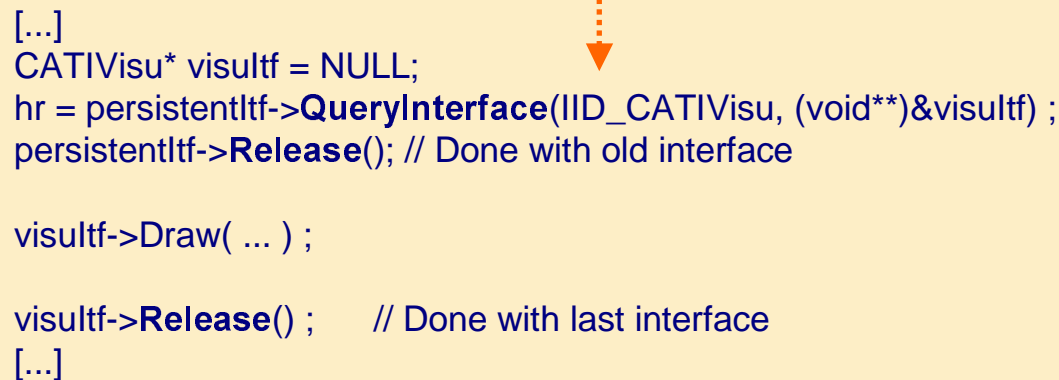
A client decrements the reference count when the interface is no more needed.



The removal of the last reference causes the deletion of the interface by ObjectModeler.

Reference Counting at work

... QueryInterface does an **AddRef()** to secure the interface pointer before returning it to caller.



```
[...]  
CATIVisu* visultf = NULL;  
hr = persistentltf->QueryInterface(IID_CATIVisu, (void**)&visultf) ;  
persistentltf->Release(); // Done with old interface  
  
visultf->Draw( ... ) ;  
  
visultf->Release() ;    // Done with last interface  
[...]
```

Remark : a pointer copy does not increase the reference count, we must call AddRef()

Factory and Reference Counting

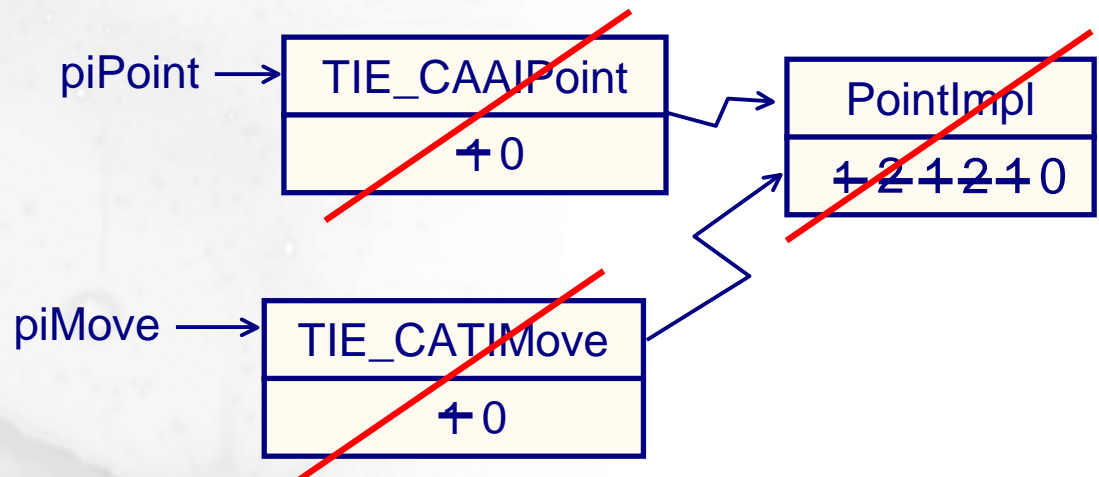
```
CAAIPoint *piPoint = factory->CreatePoint( );
```

```
└ PointImpl *pPointImpl = new PointImpl( );  
  pPointImpl->QueryInterface( IID_CAAIPoint, (void**)&piPoint);  
  pPointImpl->Release( );  
└ return piPoint;
```

```
piPoint-> QueryInterface(IID_CATIMove, (void**)&piMove);
```

```
piPoint->Release( );
```

```
piMove->Release( );
```



Reference Counting: Golden Rules



Methods returning a pointer on interface (ex: QI, factory) should always AddRef() it prior to returning



Clients receiving such a pointer should always Release() it after usage

- *Tip: write the Release() code just after the function call that created the interface pointer and « push » it with usage code*
- *Tip: when using QI(), you usually can Release() the old interface after getting the new one*



Avoid passing interface pointers around

- *If forced to do so, carefully AddRef() them*

Reference Counting: Golden Rules



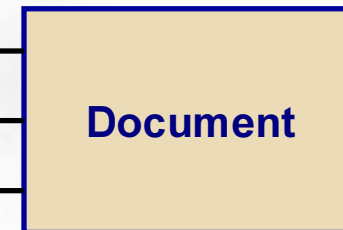
Adopt a consistent naming scheme

- *Manipulating the same object through many different interface pointer quickly gets confusing*

CATInit*
CATIPersistent*
CATIDocAlias*

piInitOnDoc
piPersistentOnDoc
piDocAliasOnDoc

CATInit○
CATIPersistent○
CATIDocAlias○



Smart Pointers (also called Handlers)

A smart pointer is a class associated to an interface, that behaves like an interface pointer, with additional automatic reference counting



Smart pointers for interface CATlxxx are of type CATlxxx_var



Being objects, they manage reference counting through code in destructor, constructor, assignment operator, etc.

Smart Pointers Arithmetic

CATIVisu_var
CATITransform_var

aCATIVisu ;
aCATITransfor ;

Operator	Example	Meaning
->	aCATIVisu->Draw()	Dereference. Access a public member of the interface. It makes the usage of interface very similar to a regular C++ pointer on implementation. Remark: do not use the "." operator with handlers.
=	aCATIVisu = aCATITransfor	Assignment. Make aCATIVisu refer to the same implementation as aCATITransfor. Since those handlers are not of the same type, it manages a downcast if left value is a subtype of right value. If not, it runs a QueryInterface on left value. Therefore the result can be a NULL handler.
()	aCATIVisu(CATITransfor)	Copy Construction. Handlers preferred way of performing a QueryInterface(). Result can therefore be a null_var handler.
==	if (aCATIVisu==aCATITransfor)	IsEqual. Test if two interface instances are dealing with the same implementation. Thanks to NULL_var which defines NULL handler value.
!=	if (aCATIVisu!=aCATITransfor)	IsDifferent. Test if two interface instances are dealing with different implementations.
!	if (!aCATIVisu)	IsNull. Test if an interface instance is actually bound to an implementation.

Smart Pointers or AddRef/Release?



Using only Smart Pointers is not possible

- *For example QueryInterface() automatically increments the reference count.*



Mixing the two can be explosive

- *Memory leaks*
- *Core dumps*



We recommend to either:

- *Stick 100% to Reference Counting*
 - *Heavier to code, but consistent*
- *Use reference counting at function boundaries (passing interfaces as arguments) and smart pointers within function's scope*
 - *Code a bit simpler,*
 - *But you need to manage the mix*

Do and Do Not



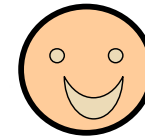
**Considering the method
CATIVisu GetIvisu();***

Don't do



```
{  
    CATIPersistent_var spPersiOnX;  
    spPersiOnX = GetIvisu();  
    spPersiOnX ->Release();  
}
```

Instead Do



```
{  
    CATIPersistent_var spPersiOnX;  
    CATIVisu *pVisuOnX = GetIvisu();  
    spPersiOnX = pVisuOnX;  
    pVisuOnX ->Release();  
}
```

**In that case the Release applies
to the CATIPersistent interface
not to the CATIVisu interface.**

Do and Do Not



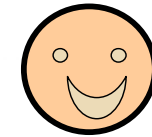
Considering the method
CATIVisu_var GetParentIvisu();

Don't do



```
{  
    CATIVisu_var spVisuOnX;  
    spVisuOnX = GetParentIvisu();  
    spVisuOnX ->Release();  
}
```

Instead Do



```
{  
    CATIVisu_var spVisuOnX;  
    spVisuOnX = GetParentIvisu();  
}
```

**In that case the return value is a
smart pointer that increments the
reference count upon creation
and decrements it upon deletion.**

Extension Mechanism

You will learn how to provide existing objects with new behaviors

- Extension
- Extension Example
- Dictionary
- Extension at work
- Extension Types

Extension

An **extension** is an object that adds **new capabilities** to an existing implementation object.



*An extension **implements** interfaces to create a component*



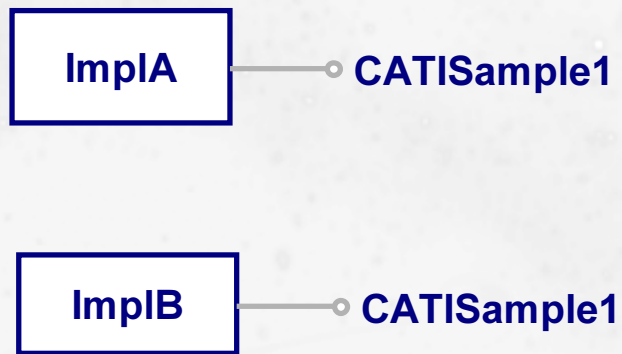
Component = Base + Extension + Interfaces



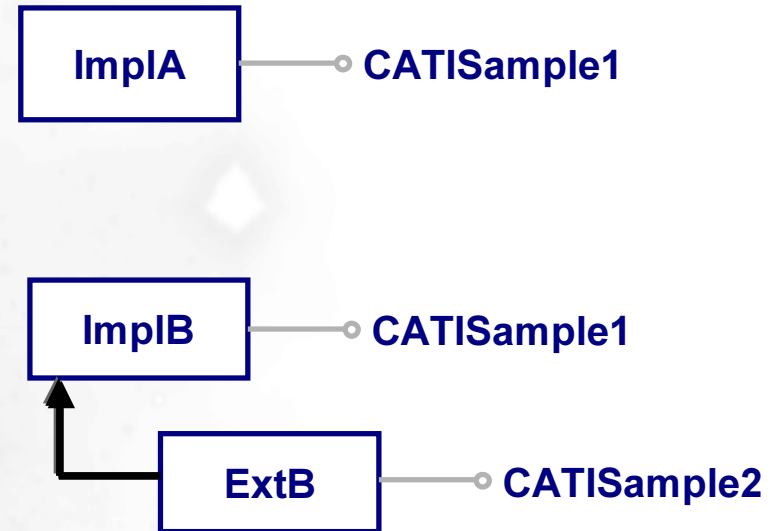
An extension can rely on capabilities exposed by its base implementation or other extensions

Extension Example

As delivered by DS



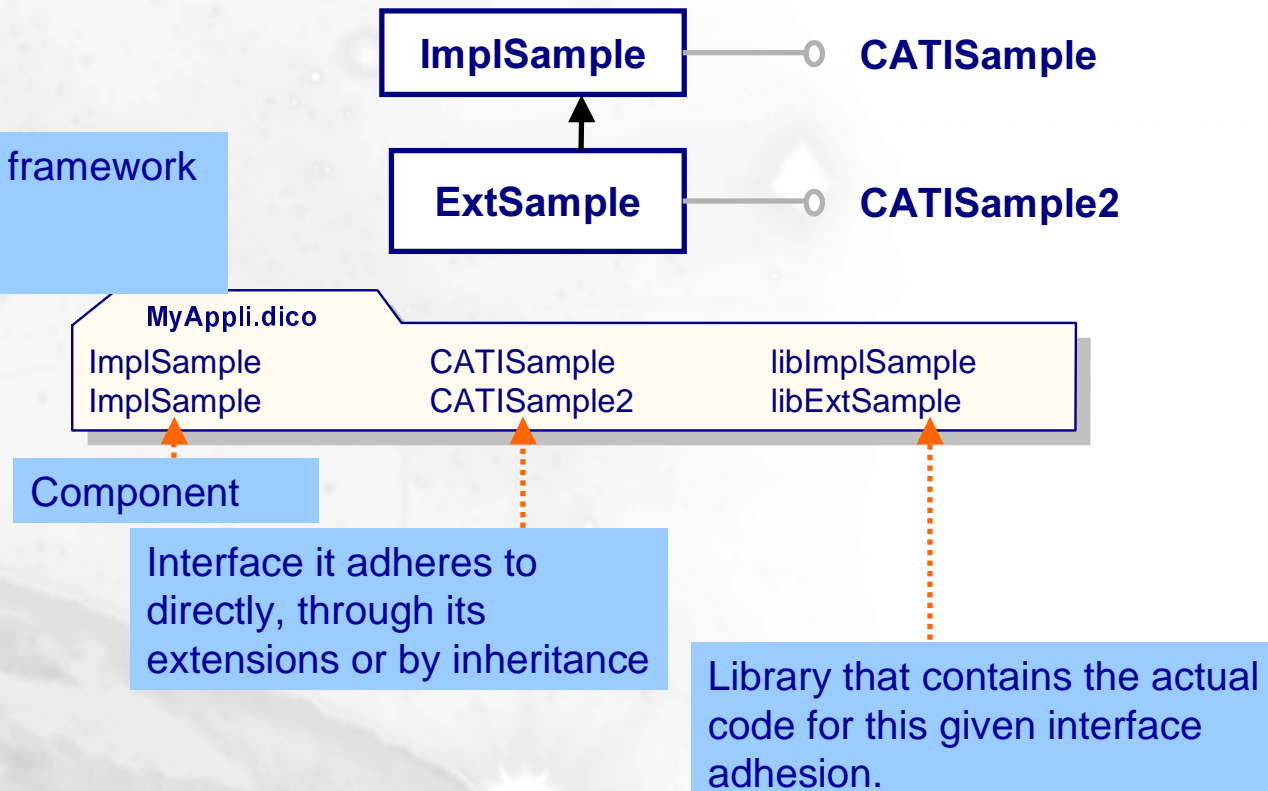
As modified by customer



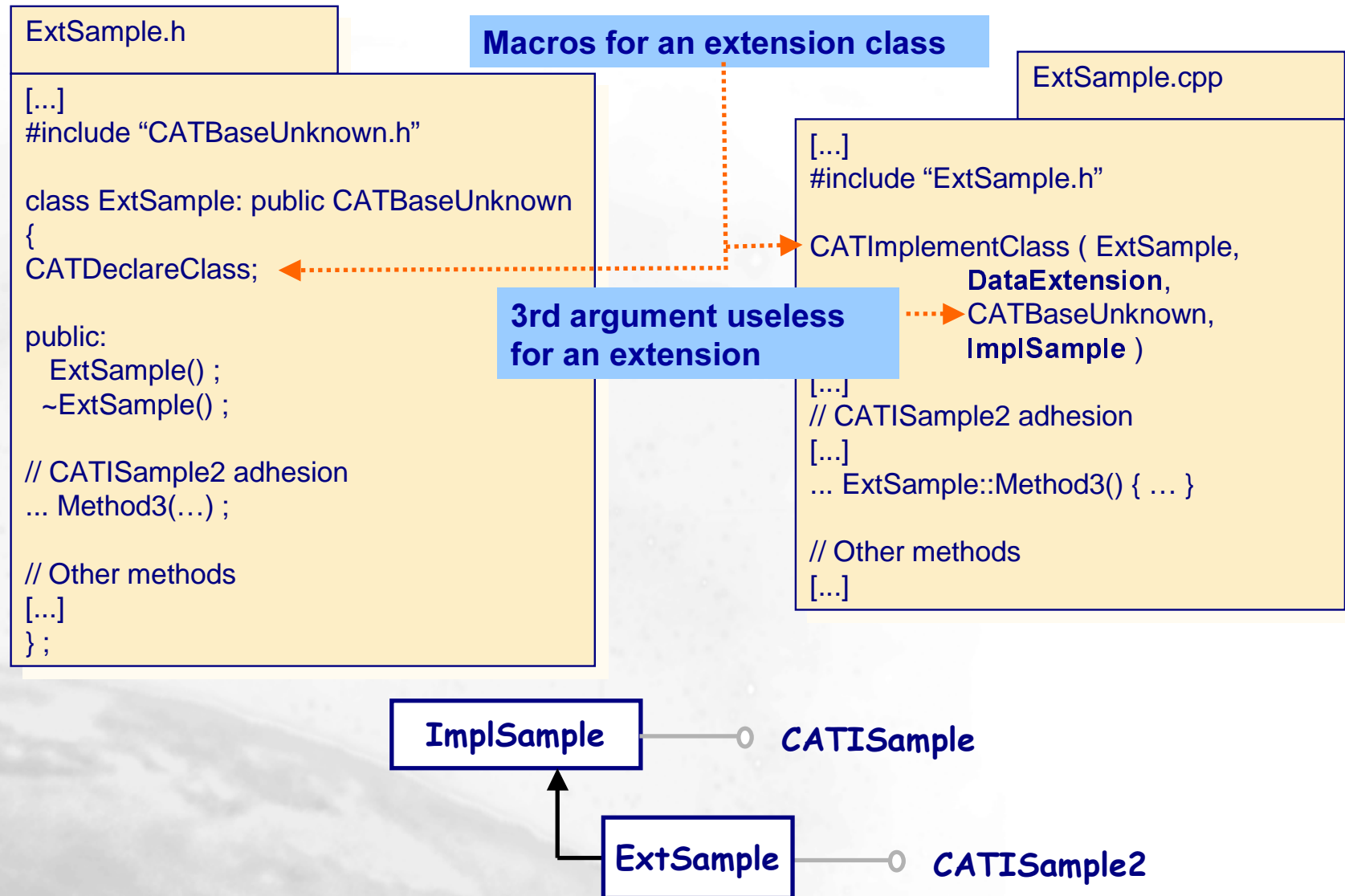
Dictionary

A dictionary is required to locate all other interfaces bound to a given implementation or its extensions.

Usually one dictionary per framework
stored under
...\CNext\code\dictionary



Extension At Work



Extension Types



Data extension

- It contains methods and data.
- *One extension instance per implementation object instance*



Code extension

- It contains only methods.
- *A single extension instance for all the implementation object instances*

Late Typing

In this lesson you will learn how to instantiate types unknown at build time

- ▣ Late Type
- ▣ Late Type at work
- ▣ Late Type Instantiation
- ▣ Late Type Inheritance

Late Type

The **late typing** is a mechanism which enables the instantiation of components by name at run time



An object is defined by a character string, its late type, and not any more by the name of the implementation class.



A late type object adheres to interfaces using the extension mechanism.



The generic object that provides this capability is CATObject.



Late typing is used for documents, containers, features ...

- *ie object types that the system must be able to instanciate in some generic code written prior to these new type definition (hence late)*

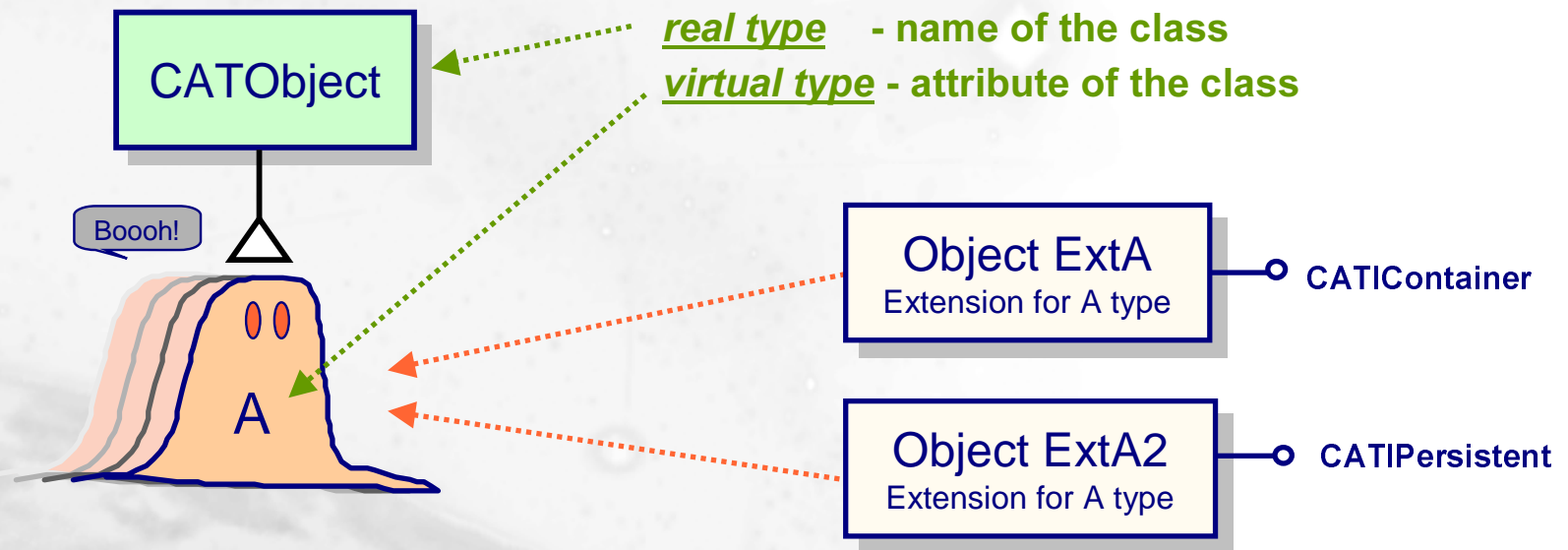
Late Type At Work



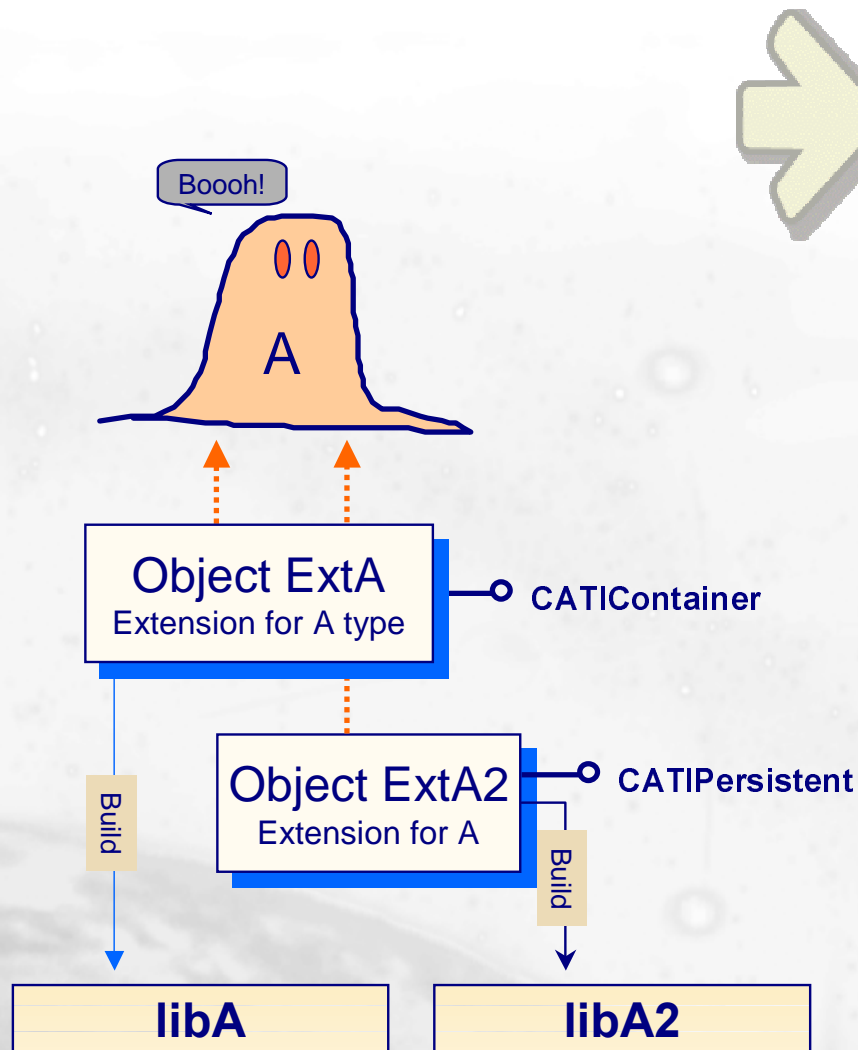
*Extension declares the type of implementation object they extend.
The extended type is specified with a string.*

In case of late type it refers to an instance of a virtual class.

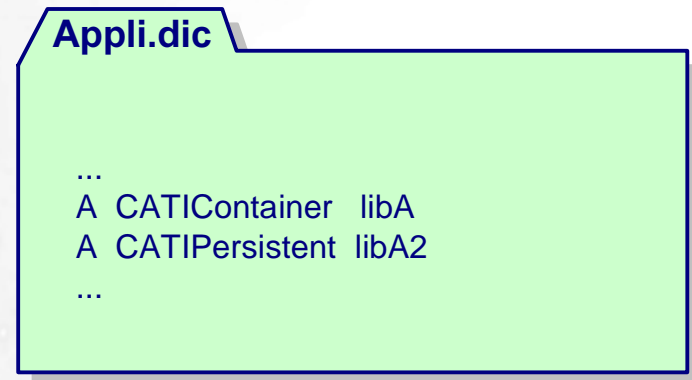
*The instance is a CATObject one but the corresponding type is
the one defined by the virtual type.*



Late Type At Work



The first column of the dictionary refers to a late type name instead of an actual class name.

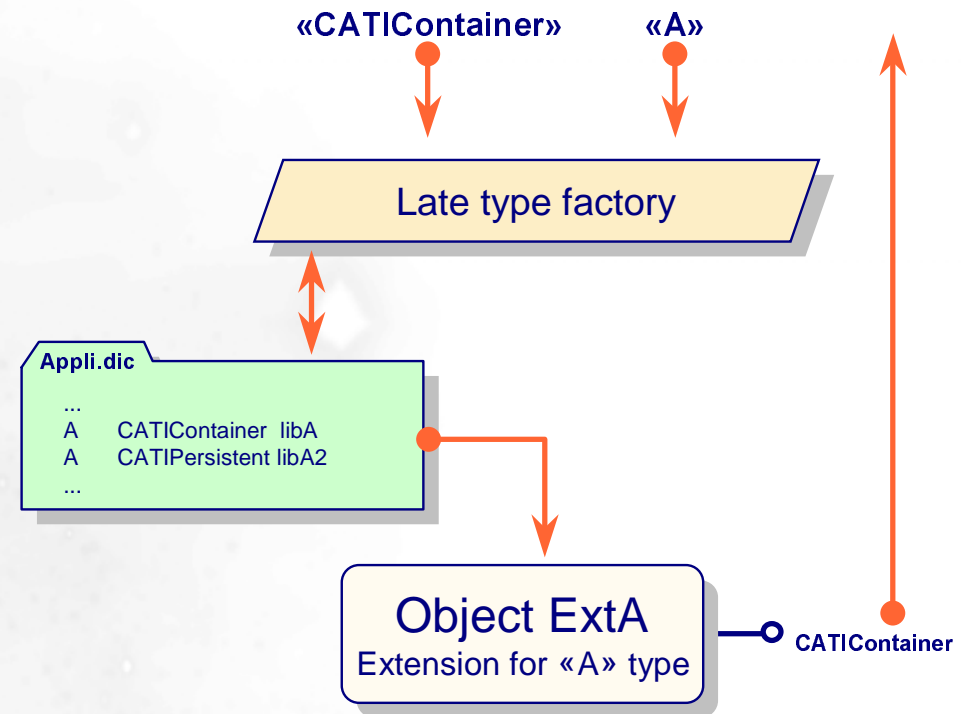


Late Type Instantiation

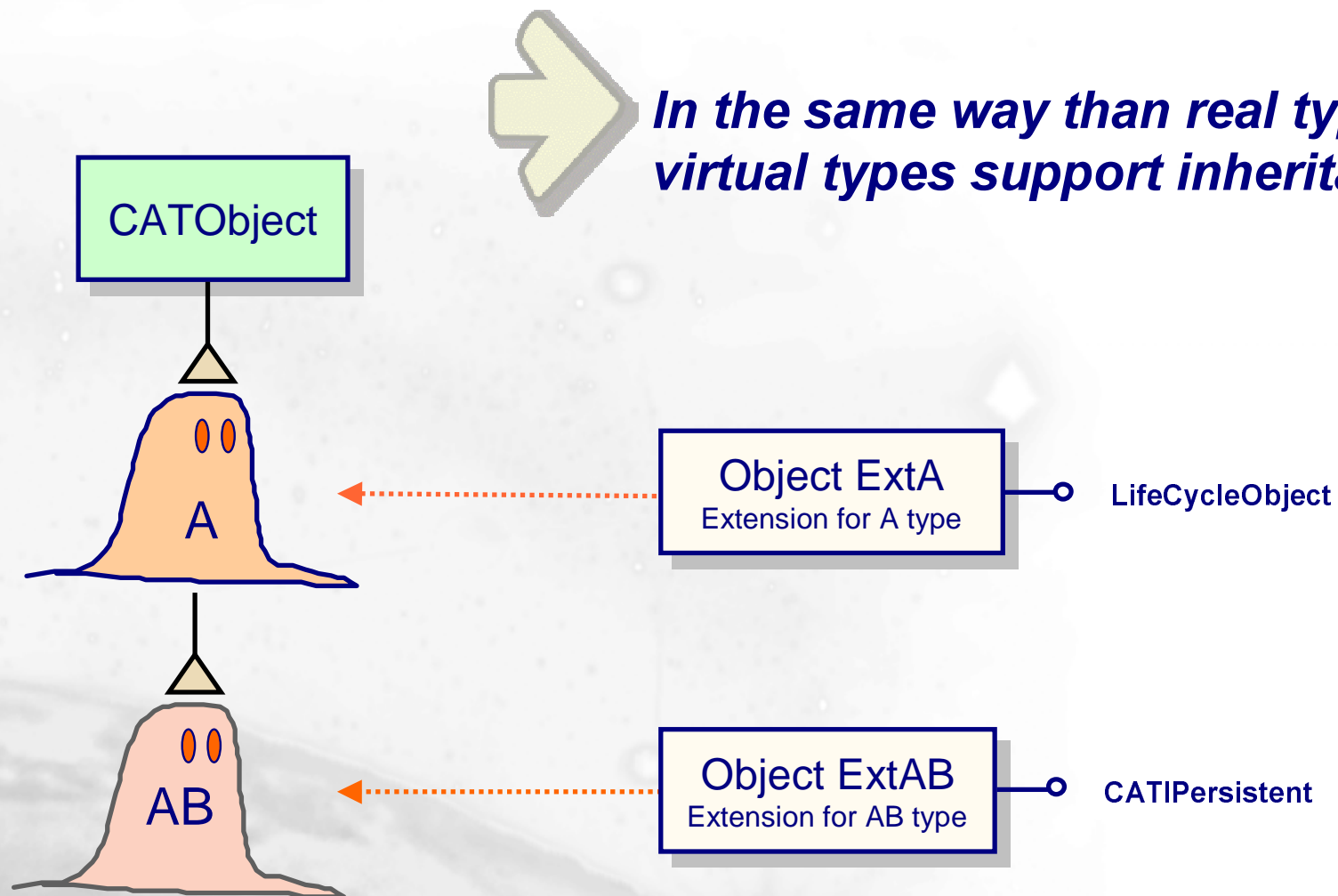
➔ *Late types are instantiated by special entities (global functions or factories).*

➔ *The factory uses the dictionary to find out if the required interface is supported by the late type.*

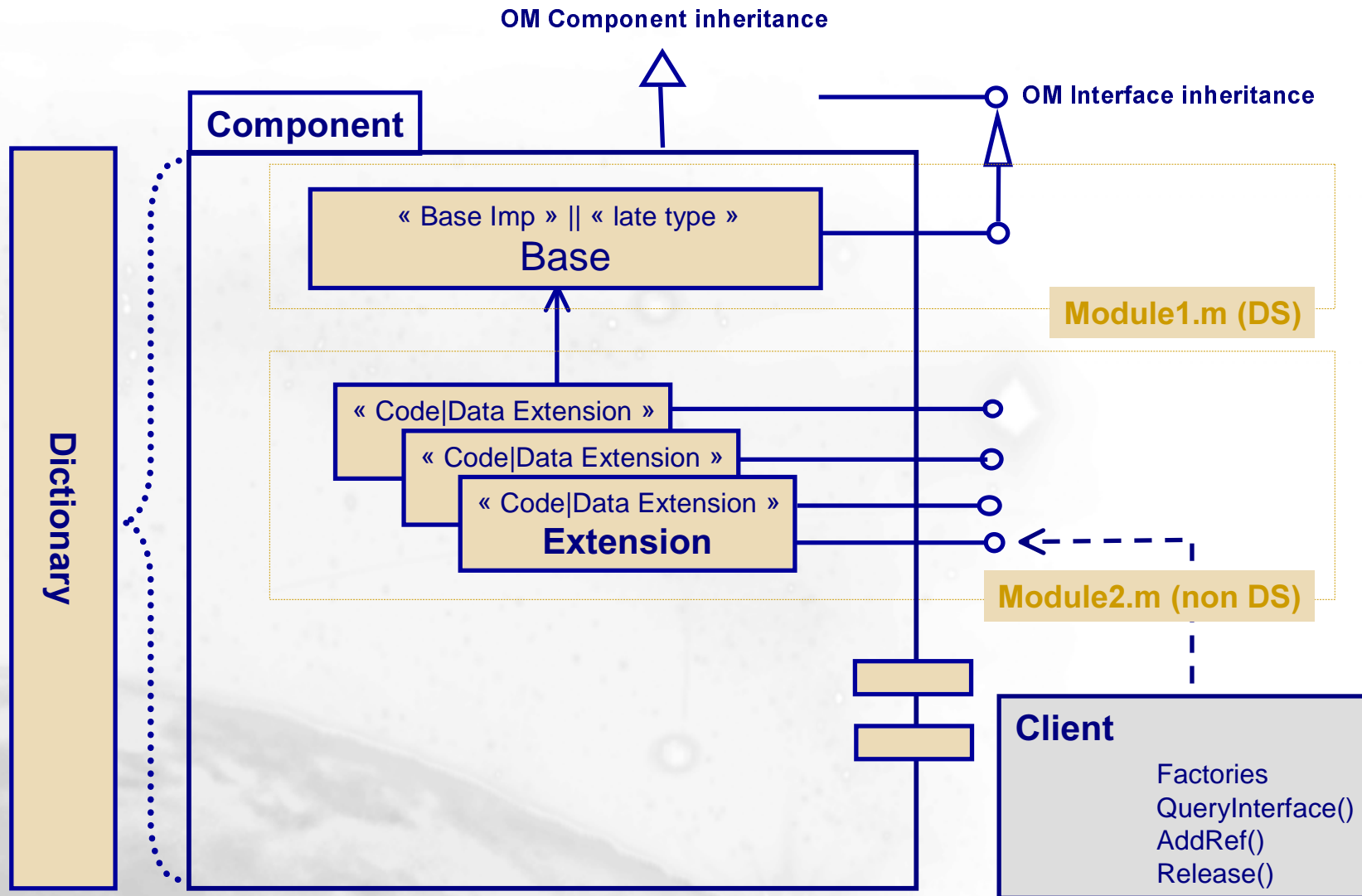
➔ *Thanks to libA2 a new extension is created. An instance of the required interface is retrieved and passed back to the requester.*



Late Type Inheritance



Additional Information : Epilog



To Sum Up ...

In this Course you have seen...

- **Objects handling with Interfaces**
- **The Implementation and Extension mechanism**
- **The Late Typing**

CAA V5 ObjectSpecsModeler

In this course you will learn how to create your objects in CATIA V5 infrastructure and how to define their behaviors

- Introduction to Specification Modeling
- Object Specs Modeler Objectives
- About Features
- Programming Tasks
- About Features Extensions
- Programming Tasks
- About Providers

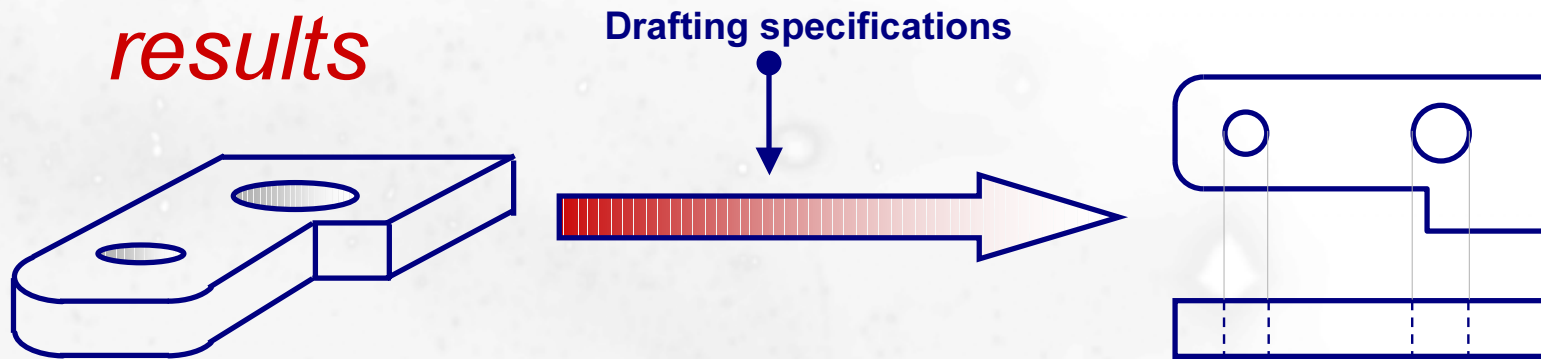
Introduction to Specification Modeling

In this lesson you will learn the need in Specification/Result

- ▣ Accelerating the Art-To-Part Process
- ▣ CATIA Application Evolutions
- ▣ Needs For Next Generation Products
- ▣ CATIA Specification Modeler Objectives

Accelerating the Art-To-Part process

- Capturing *specifications* and Generating *results*



Benefits:

- *Results are generated faster*
- *Company rules and know-how are captured*
- *Result is explainable*
- *Designers do design, not clerical tasks*

CATIA Application Evolutions

First Generation CATIA V1/V2

- Drafting
- Surface Modeling
- ...

- ➔ *User produced results rather than specifications*
- ➔ *Changes upwards in the spec/result tree cause big reworks downstream*

Second Generation CATIA V3/V4

- CSG Solid
- Generative Drafting
- GSM
- Sheet Metal Aerospace
- ...

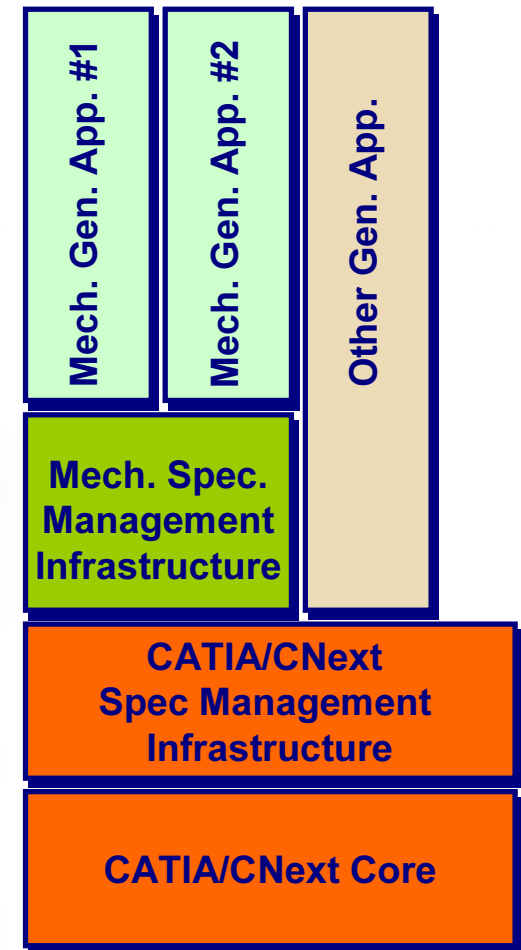
- ➔ *First Specification editors*
- ➔ *First generation applications flaws are removed **BUT...***
- ➔ *Every application defines its own spec/result paradigm*
- ➔ *Applications are not all spec/result oriented*

Needs For Next Generation Products



Provide a general specification/result management platform within the CATIA infrastructure

- *Offering common services to all applications willing to go the « generative way »*
- *Open for customization by those applications*
- *In order to*
 - *Speed up the development of such applications*
 - *Deploy a consistent and pervasive approach for spec/result throughout CATIA*



ObjectSpecsModeler Objectives



Provide an infrastructure for Specification/Result management, and therefore Associativity

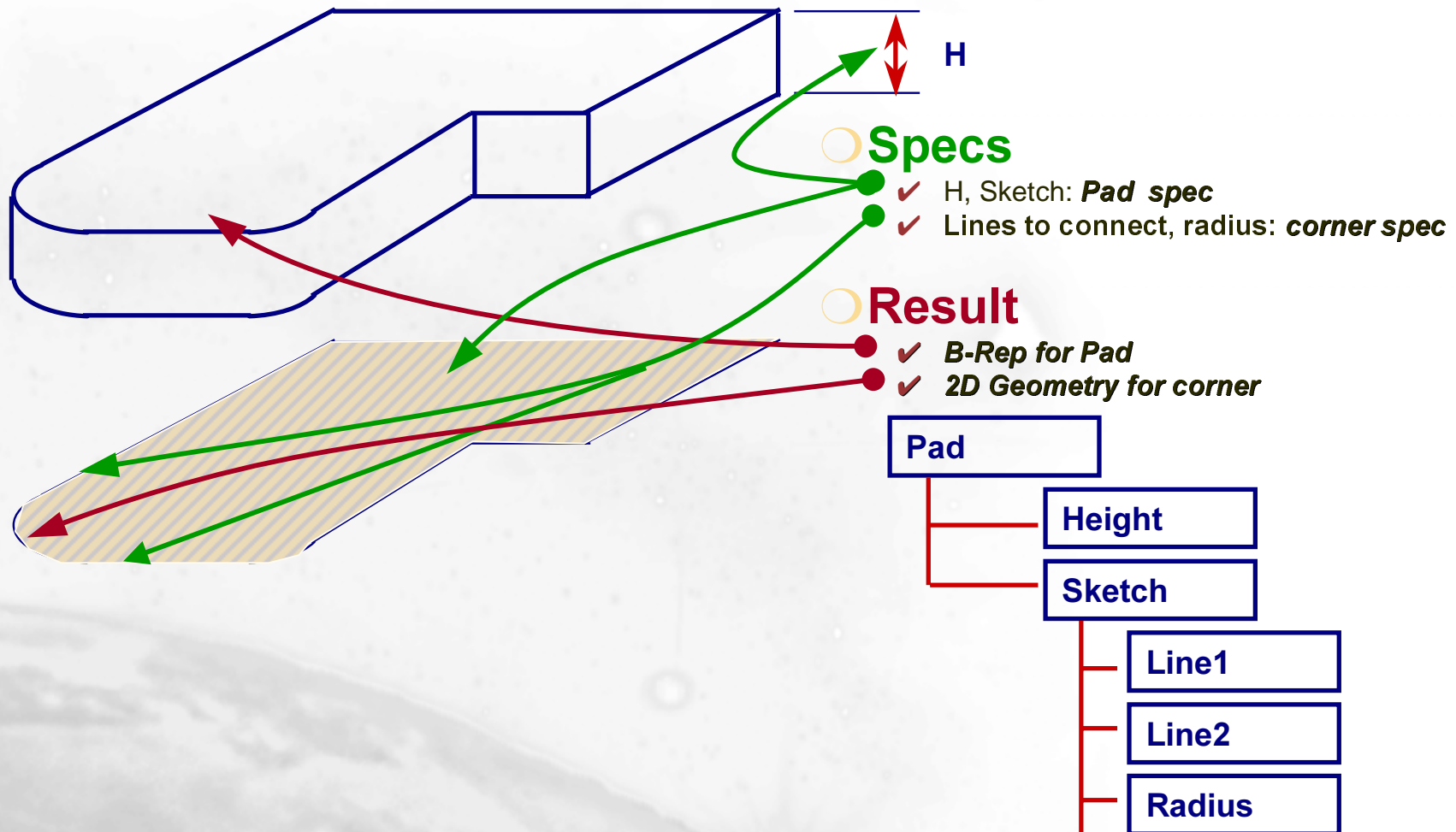
- *Capability to link specifications by aggregation or reference*
- *Generic mechanism for change propagation*
- *Persistency*



Model should be extendable at runtime

- *with user defined specifications*
- *with new Dassault Systèmes specifications*

Specification and Result Management



Need for a Versatile Update Mechanism



There is a need for a generic update mechanism.



This mechanism should be transparent enough to minimize coding effort,



But yet customizable to let application developers define their own update policy:

- *Manual or automatic,*
- *Local or global.*

Need for User Defined Specifications



That can be defined at run time



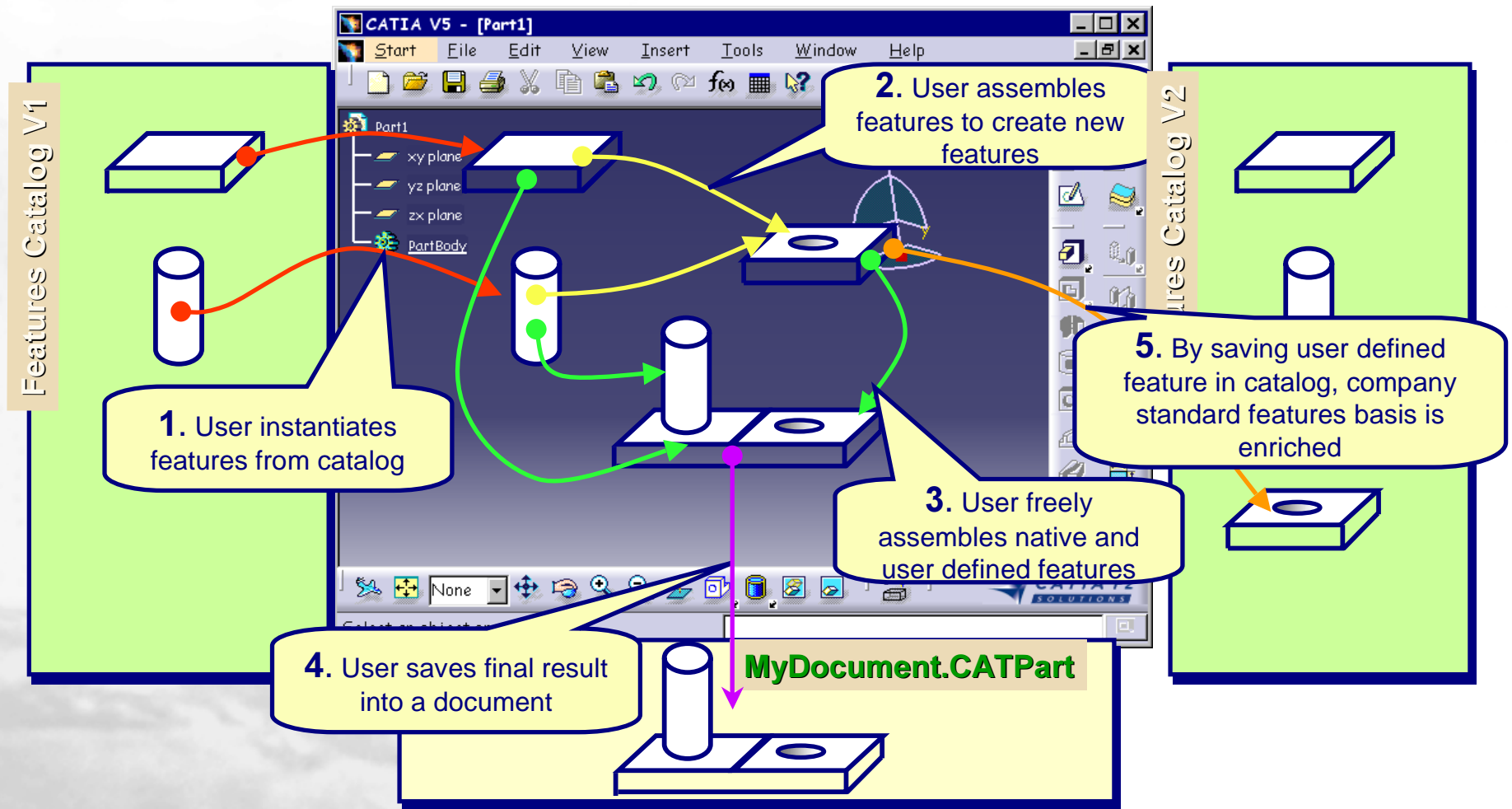
That can be made persistent for future usage

- *Specification catalogs*



That are treated as peers by CATIA-defined specs

Need for User Defined Specifications



ObjectSpecsModeler Principles

In this lesson you will learn the Data modeler and model of CATIA V5

- ▣ **Solution Provided by the Spec Modeler**
- ▣ **A Data Modeler with Embedded Services**
- ▣ **A Generic Update Mechanism is Provided**
- ▣ **A Catalog Based Data Model**
- ▣ **Prototype / Instance**

Solution Provided by the Spec Modeler



A Data Modeler with embedded services

- *For a Feature based application*
- *Along with a mechanism to manage updates*



A Data model that is stored in a catalog

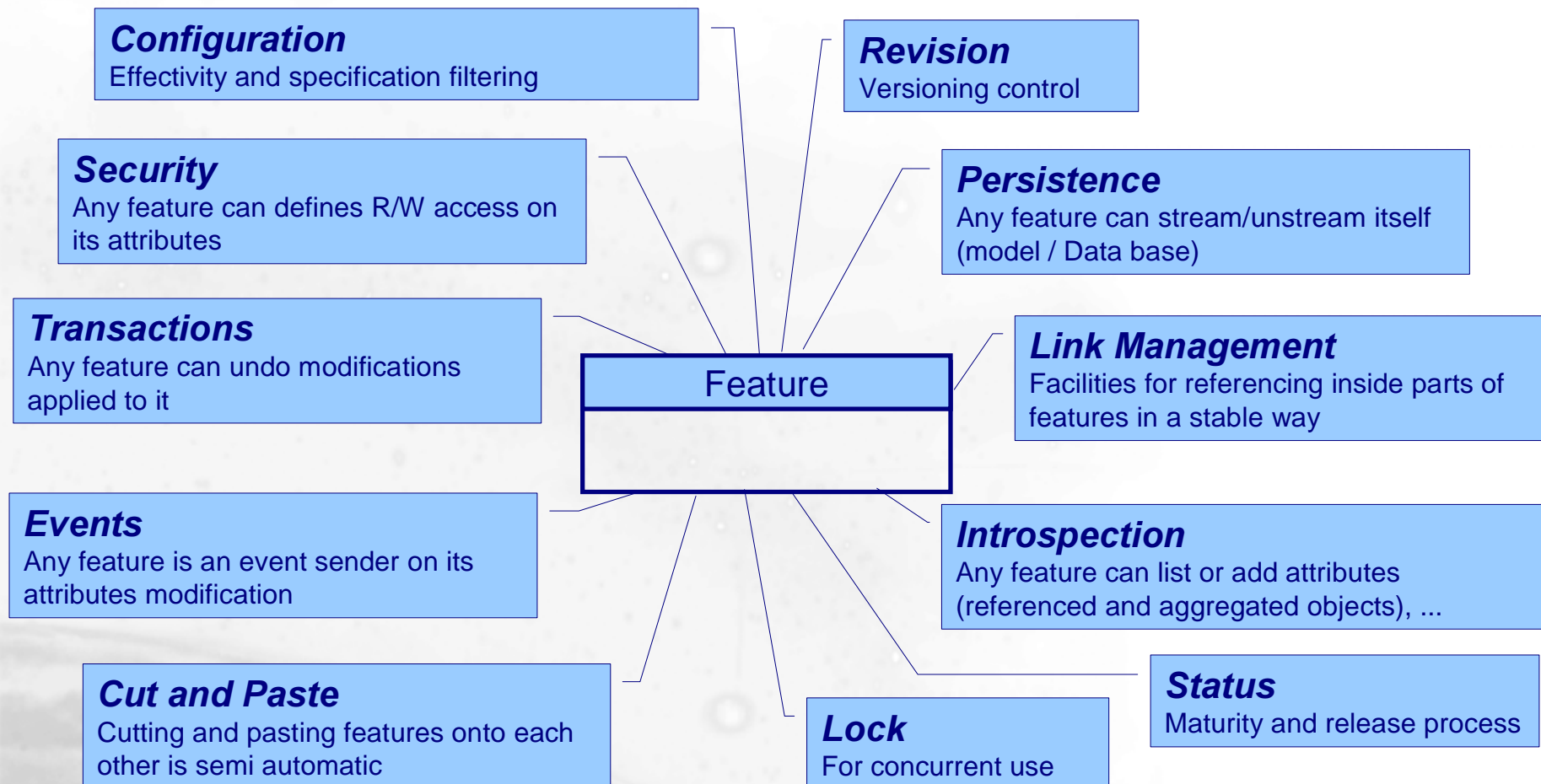
- *No hard coded specification*
- *Extensibility, possibility for user defined specification*
- *Data model can be modified, even at runtime*
- *Introspection*



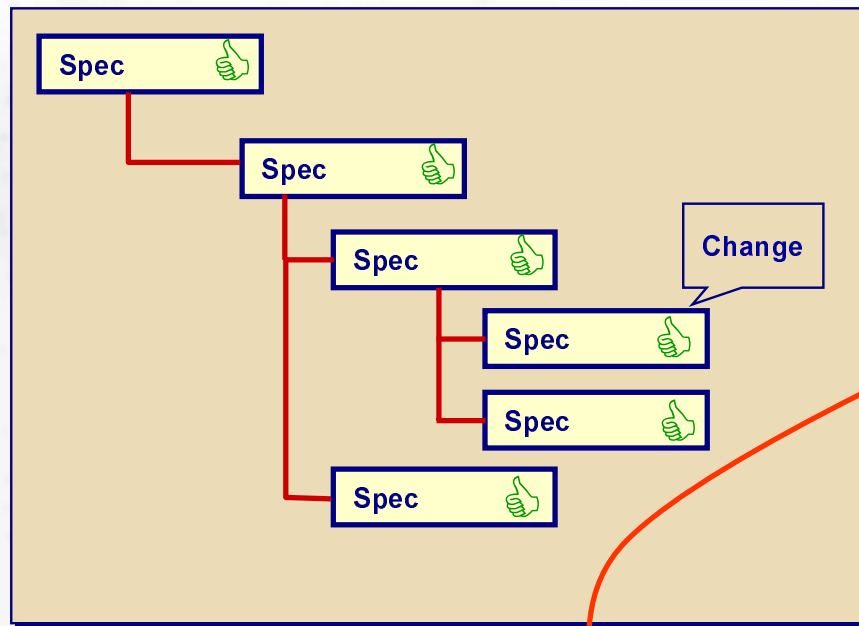
A Prototype/Instance object model

- *To allow dynamic instantiation*

A Data Modeler with Embedded Services

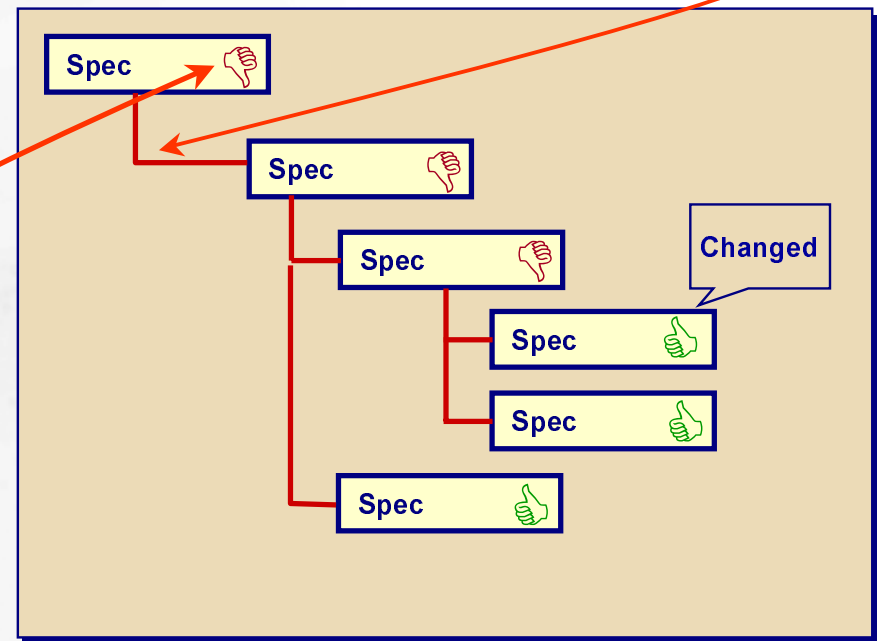


A Generic Update Mechanism is Provided



...while deciding when and how responding to change request is under the responsibility of the application

Propagating change requests across the spec tree is under the responsibility of the spec modeler infrastructure...



A Catalog Based Data Model



The Data model is defined in a separate document

- *A feature catalog*

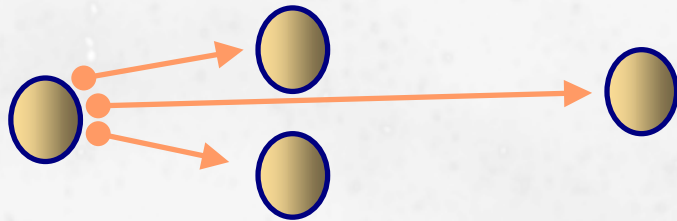


This approach has the following advantages:

- *Extensibility, a feature can be extended by inheritance with new attributes later on with a complete compatibility with existing instance.*
- *Maintainability, the data model is not hard coded, therefore implementation can be modified.*
- *Introspection, a feature knows its attributes, and its super type (not the case in C++).*

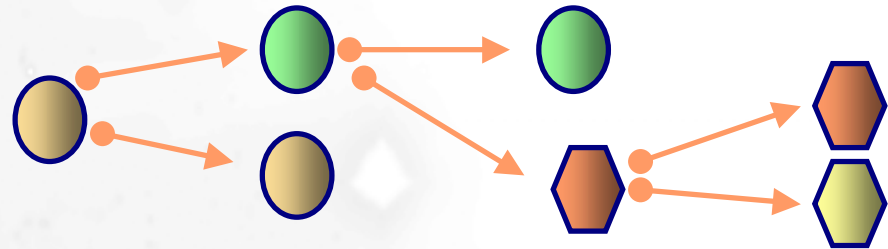
Prototype / Instance

Reference/Instance C++



- *All instances for a given reference are alike*
- *New instances can be obtained only from the reference*

Prototype/Instance SpecsModeler



- *Instances can be made different from the reference*
- *Instances can in turn be used as new references*
- *Every object is instance & reference at the same time (hence « prototype »)*
- *Instances and prototypes can be synchronized (attributes structure) when needed*

About Features

In this lesson you will learn the CATIA V5 Features creation and management

- ▣ What is a CAAV5 Feature
- ▣ Feature Update Mechanism
- ▣ Feature Attributes
- ▣ Feature Catalog and Instantiation
- ▣ Feature and Late Typing
- ▣ Feature and Persistency
- ▣ Feature Interfaces
- ▣ Summary

CAA V5 Features



A feature is the basic object for specification modeling.

- A feature stores specifications and produces one or several results through an update mechanism***
- It owns some Attributes***
- It follows the prototype/instance pattern***
 - The prototype is the skeleton of the Feature, its data description***
- It is a late type object***
- It is persistent***

Specification and Result Management



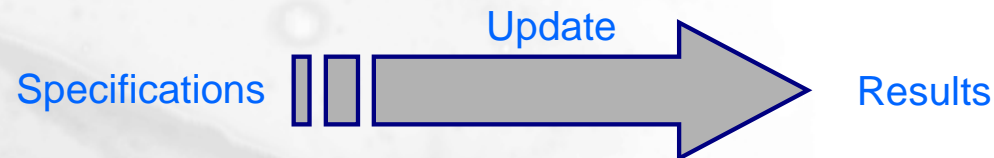
A feature owns some specifications and generates results through an update mechanism.

- Feature's update is triggered by a modification in the specifications.*
- A specification can be a string, a real, an integer or another feature.*
- A result can be anything from features to geometry.*



Specification = Input data

Result = Output data

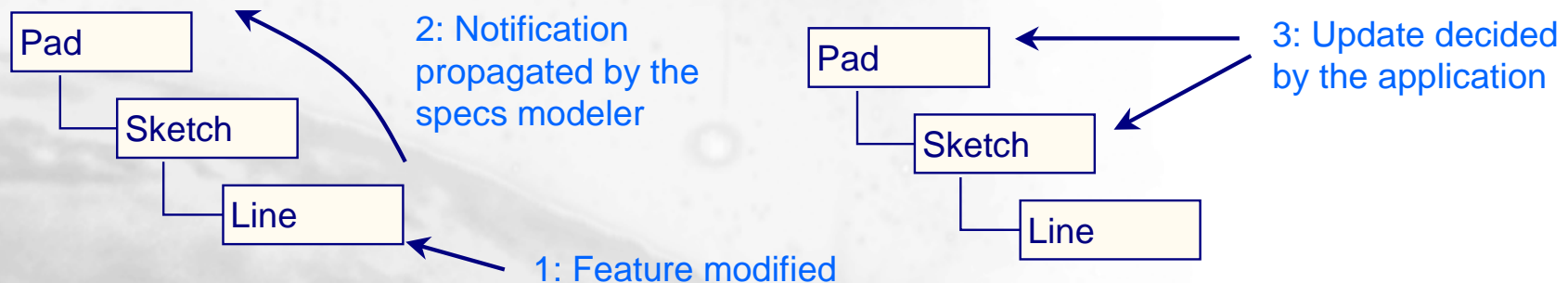


Feature Update Mechanism



The Specs Modeler provides the infrastructure to manage updates.

- It automatically propagates notification of change across the specification tree, but it does not trigger the update itself.***
- Deciding if an object must be updated or not is the responsibility of the application***



Feature Attributes



Specifications and results are stored in the feature's attributes.

- An attribute is similar to a class member in C++***
- An attribute can be a double, an integer, or another feature.***
- An attribute has a quality (input, output or neutral)***
 - Input attributes store Specifications***
 - Output attributes store Results generated from the Specifications***
 - Other properties are stored in Neutral attributes.***

Attributes and Update

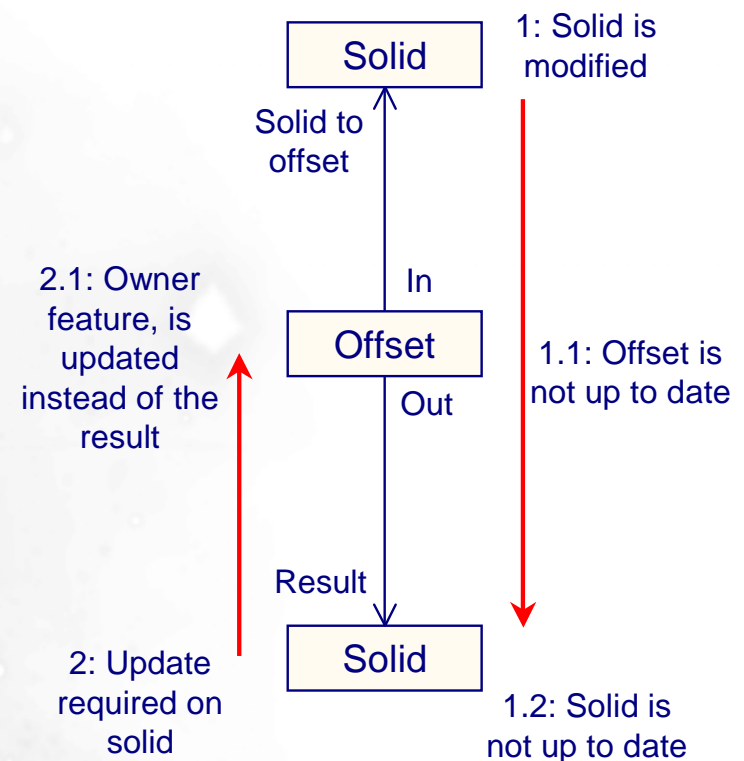
➡ When a feature is modified (1)

- Features that take this feature as an input are marked not up to date (1.1)
- Features that are output of this feature are marked not up to date (1.2)

➡ When a feature must be updated (2)

- If this feature is an output of another feature, then its owner is updated instead (2.1)
- Input features are updated first

➡ In both case neutral attributes are ignored.



Feature Catalog



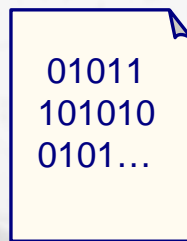
Feature definitions are stored in a catalog file

- A feature definition (skeleton) is called a Startup***
- There can be as many catalog file as needed.***
 - User catalog files end with .CATfct*
 - DS catalog files end with .feat*

Steps to create a new Startup:

1: Create or
upgrade catalog

3: Save catalog



Catalog



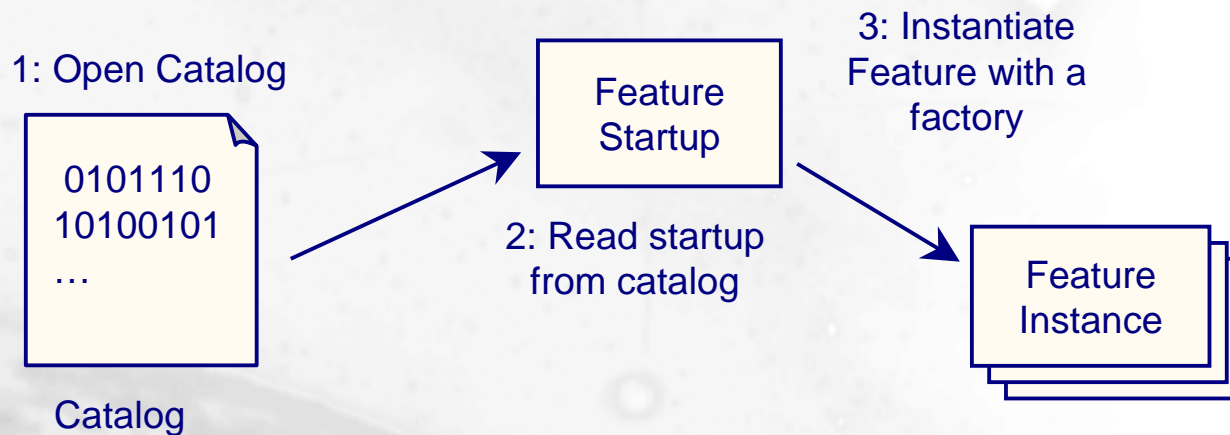
2: Create startup in
catalog

Feature Instantiation



Instantiation follows the Prototype/Instance Pattern

- A feature is instantiated from its startup.***
- The startup must be retrieved from the catalog.***
- The instance will be an exact copy of the startup and will retain any attributes with their initial default value.***



Feature and Late Typing



A feature is a late type:

- ***Behaviors are provided through the extension mechanism (an Object Modeler service).***
- ***Inheritance between feature is allowed***
 - ***New feature will inherit both behaviors and attributes of its super type***

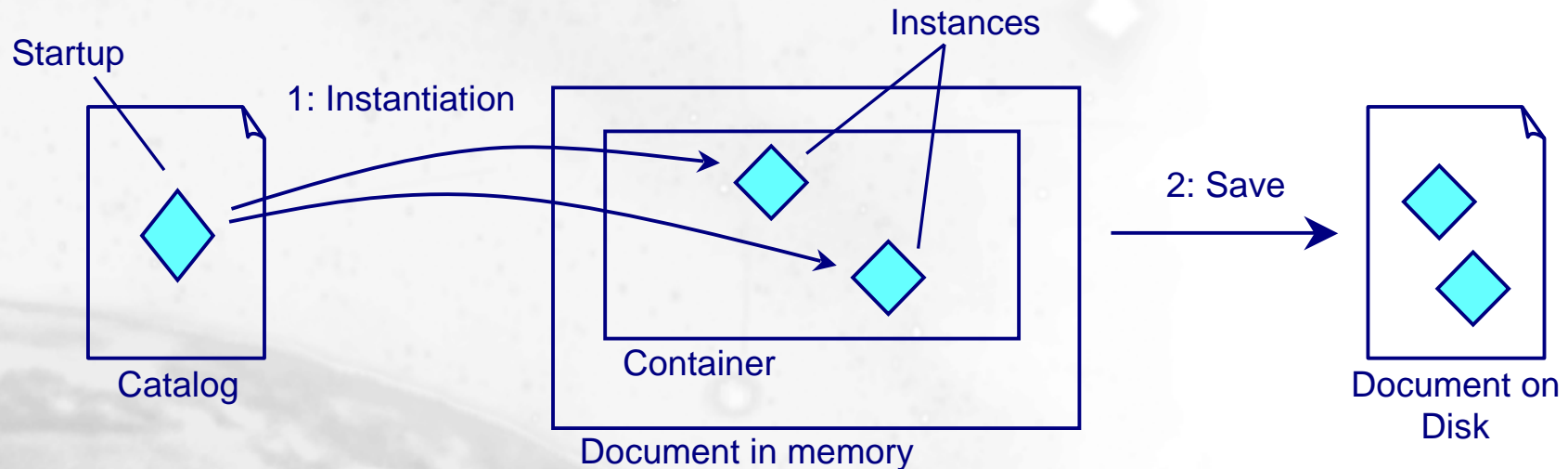


Feature and Persistency



***To be persistent a feature must be created in a container.
This container belongs to a document.***

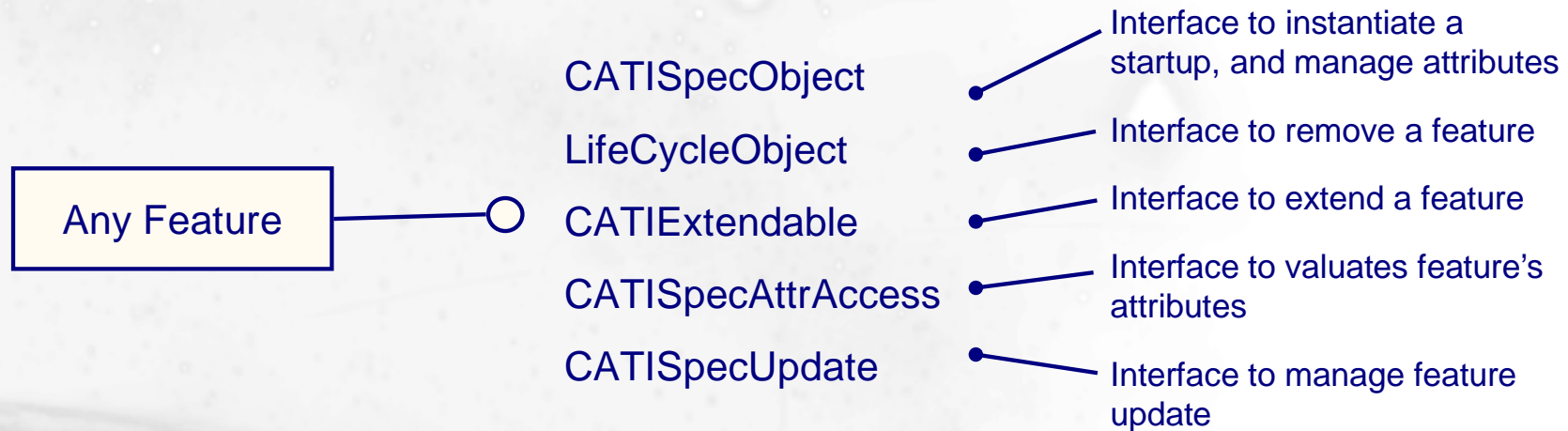
- Saving the document will in turn save the feature.***



Feature Instances



Every feature is an instance of a CATSpecObject
– It implements several DS interfaces:



Feature Interfaces Migration



From R8, feature interfaces will undergo some major changes.

- *CATISpecObject interface is deprecated and will be replaced by the following interfaces:*
 - *CATISpecBase for feature instantiation*
 - *CATISpecAttrManager for attribute management*

- *Other deprecated interfaces are:*
 - *CATISpecAttrAccess (attribute valuation) will be replaced by CATISpecAttrValue*
 - *CATISpecAttrKey (attribute index for fast access) will be replaced by CATISpecKey*

Feature Interfaces (V5R8)



At the end of the migration, feature interfaces will be:

Any Feature

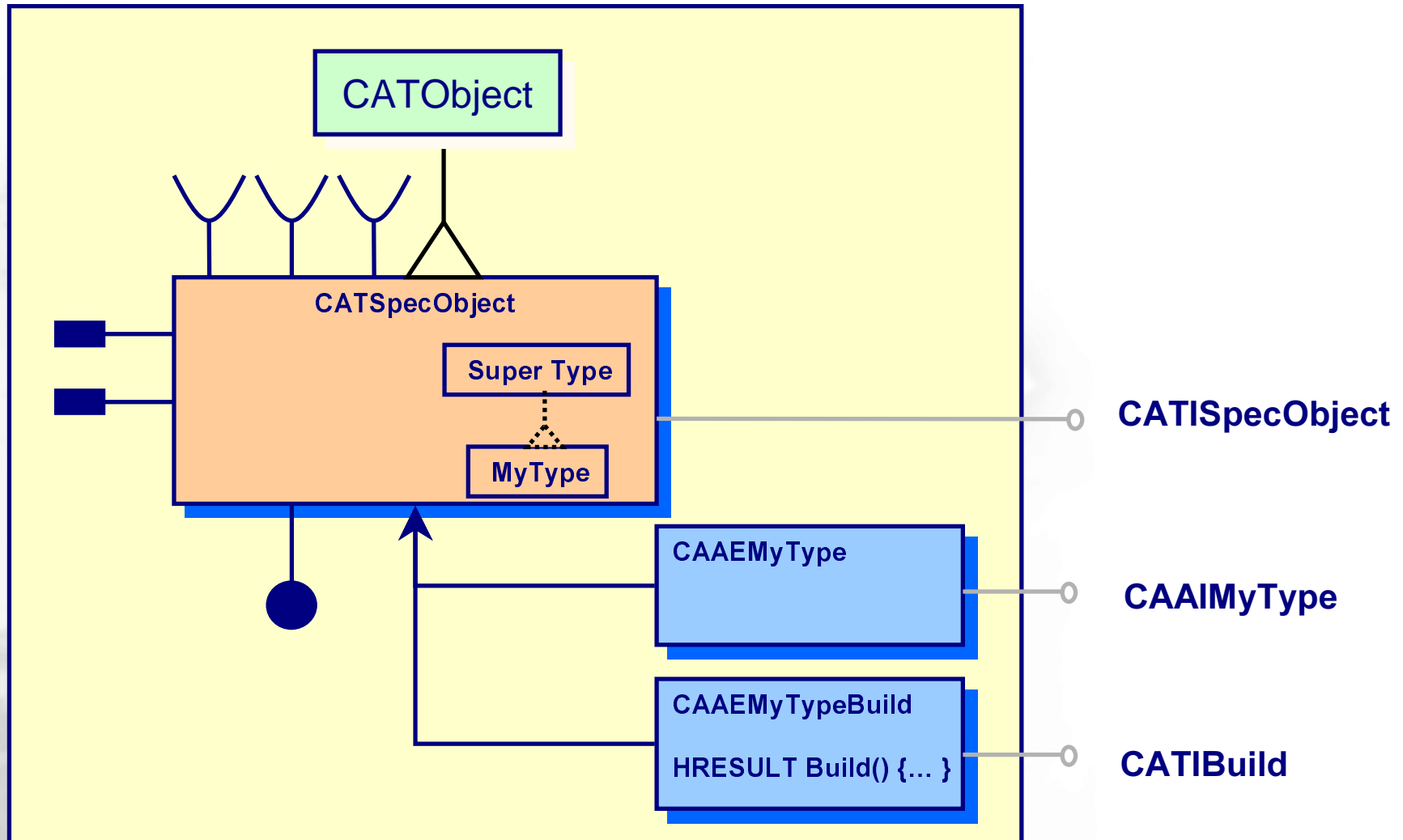


CATISpecBase
LifeCycleObject
CAT Extendable
CATISpecAttrManager
CATISpecAttrValue
CATISpecUpdate

- Interface to instantiate a startup
- Interface to remove a feature
- Interface to extend a feature
- Interface to manage feature's attributes
- Interface to valuates feature's attributes
- Interface to manage feature update

About Features

Summary



Programming Tasks

In this lesson you will learn how to implement Features

- **Creation of a New Feature**
- **Feature Catalog management**

Process to define a new feature



Design my Start-up



Define the reference data structure

- *Create or upgrade a Catalog*
- *Create the Startup*
- *Define the Startup attributes*
- *Save the Catalog*



Implement the different behaviors

- *Provide the implementations for the interfaces to be supported by the feature through Object Modeler extension(s)*
- *If needed, provide the Build mechanism*



Create instances

- *Clone the startup*
- *Set the instance attributes*

Parallel between regular C++ and ObjectSpecsModeler

C++

Data Structure defined in a header file

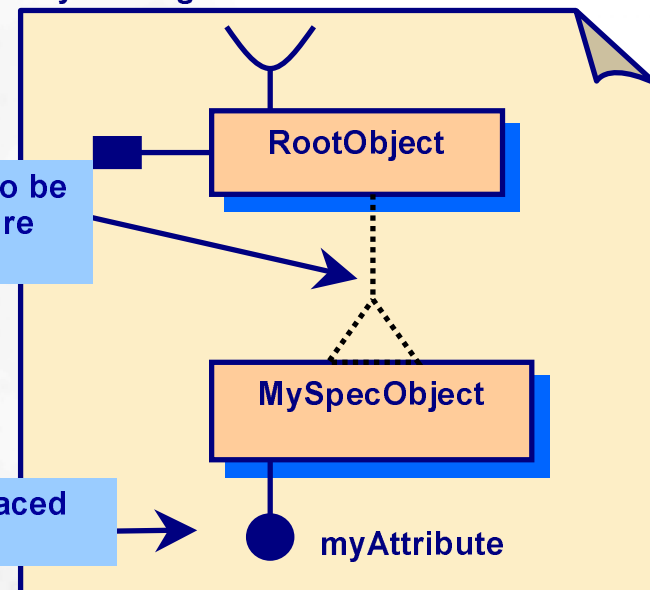
MySpecObject.h

```
Class MySpecObject: public RootObject
{
public:
    MySpecObject();
    MySpecObject(double value);
    ~MySpecObject();
//
//
private:
    double _myValue;
};
```

ObjectSpecsModeler

Data Structure defined by program and stored in a feature catalog

MyCatalog.CATfct



C++ inheritance to be replaced by feature inheritance

Data members replaced by attributes

Parallel between regular C++ and ObjectSpecsModeler

C++

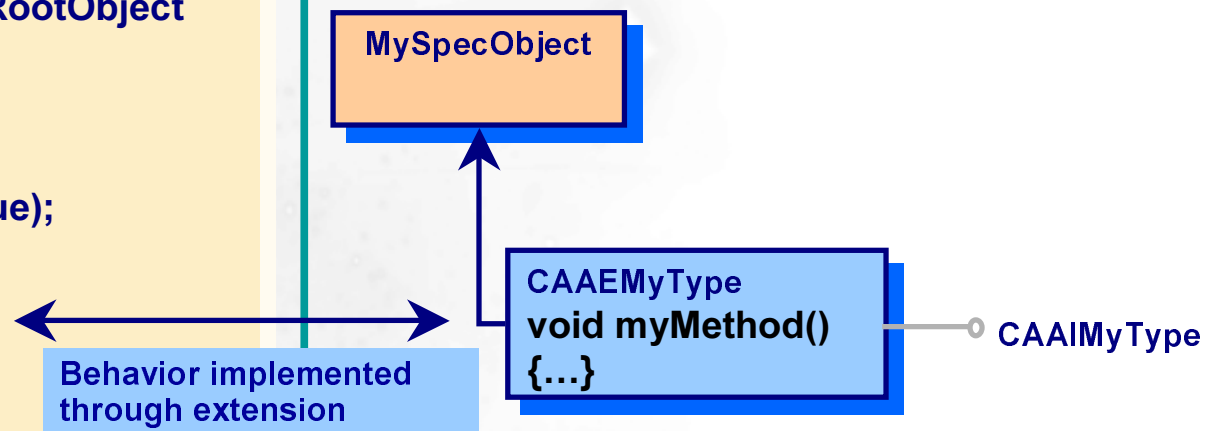
Methods are defined in a header file and implemented in a cpp file

MySpecObject.h

```
Class MySpecObject: public RootObject
{
public:
    MySpecObject();
    MySpecObject(double value);
    ~MySpecObject();
//
    void myMethod();
//
private:
    double _myValue;
};
```

ObjectSpecsModeler

Supported interfaces are defined in a dictionary and implemented by an extension



Parallel between regular C++ and ObjectSpecsModeler

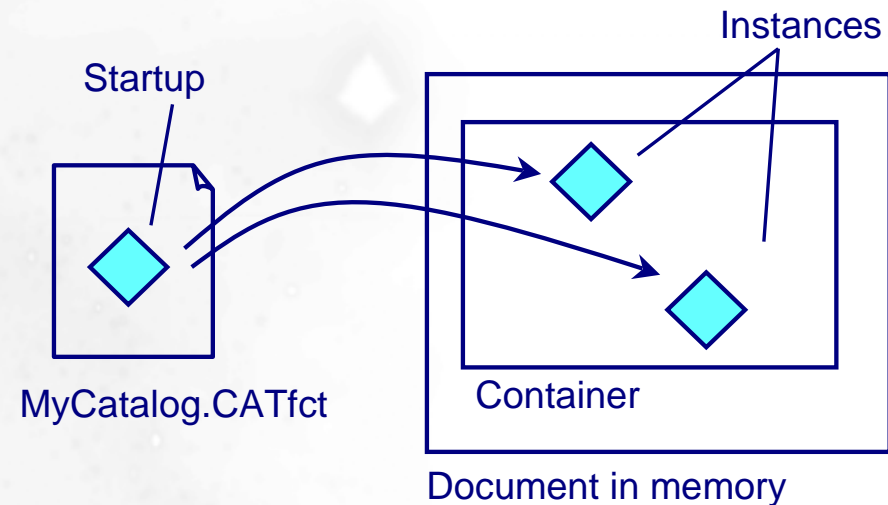
C++

Instantiation through the class constructor and the new operator

```
MyObject * myInstance = NULL;  
myInstance = new MyObject(10.0);
```

ObjectSpecsModeler

Instantiation from the startup retrieved in the feature catalog, and then attribute valuation



Catalog Definition



Create a new catalog

- ***Suffix: CATfct***
- ***Must be located in the CNext\resources\graphic directory of your framework***
- ***Use the CreateCatalog method defined in header file CATCatalogFactoryServices.h***

```
CATUnicodeString catalogName = "MyCatalog.CATfct";  
CATICatalog* piMyCatalog = NULL;  
HRESULT rc = ::CreateCatalog( &catalogName, &piMyCatalog);  
// Set a client identification for the catalog.  
CATUnicodeString clientId = "...";  
rc = myCatalog -> SetClientId(&clientId);
```

A specific identifier that allow access to your catalog.

Catalog Definition



Or upgrade an existing catalog

- ***Must be a user catalog***
- ***Use the UpgradeCatalog method defined in header file CATCatalogFactoryServices.h***
 - ***The client ID is needed to open the catalog***

```
CATUnicodeString catalogName = "MyCatalog.CATfct";  
CATICatalog* piMyCatalog = NULL;  
CATUnicodeString clientId = "...";  
HRESULT rc = ::UpgradeCatalog( &catalogName, &piMyCatalog, &clientId);
```

Catalog Definition



Create the startups

- *See next slide*



Add Attributes to your startups (Next Slide)



Save the catalog

- *Use the SaveCatalog method defined in header file CATCatalogFactoryServices.h*

```
CATUnicodeString storageName = ...;  
HRESULT rc = ::SaveCatalog ( &piCatalog , &storagePathName );
```

Startup Definition



A startup is defined by:

- ***its name (instance name)***
 - *CATUnicodeString*
- ***its late type***
 - *CATUnicodeString*
- ***its super type (optional, the late type of the startup it OM-derives from)***
 - *CATUnicodeString*
 - *The new startup will inherit all the attributes and behaviors defined at the level of the super type startup.*

```
CATBaseUnknown * myStartUp = NULL;  
HRESULT rc = myCatalog->CreateSUInCatalog(    &myStartUp,  
                                              &myStartUpName, &myStartUpType,  
                                              "CATISpecObject",  
                                              &myStartUpSuperType );
```

Startup Definition



There are three different cases

- ***Creation of a startup from scratch***
 - *Use CATICatalog::CreateSUInCatalog without super type.*
- ***Creation of a startup that derive from a user startup***
 - *If the original catalog is not already open use AccessCatalog to open it in read only mode.*
 - *Then use CATICatalog::CreateSUInCatalog.*
- ***Creation of a startup that derive from a DS startup***
 - *In the current version, open the DS catalog with OpenCatalog then call CATICatalog::CreateSUInCatalog.*
 - *Later, you will use a specific factory to create a derived startup.*

Feature Attribute Definition



Name (*CATUnicodeString*)



Type (*C++ enumeration value*)

- **A simple type**
 - *tk_string / tk_double / tk_boolean / tk_octet / tk_integer*
- **A feature type**
 - *tk_specobject*: a reference to another feature
 - *tk_component*: the feature is aggregated
- **a list of any other types:**
 - *tk_list*,
- **A more general object type:**
 - *tk_external*: a reference to an object that implements the *CATILinkableObject* interface



Quality: *defines the attribute role in the build process and update mechanism*

- **sp_OUT:** *output data*
- **sp_IN:** *input data for the Build process*
- **sp_NEUTRAL:** *attribute ignored by the update mechanism*

Adding Attributes to the Startup



Get a pointer to CATISpecObject on the startup:

```
CATISpecObject * myStartUpSpec = NULL;  
rc = myStartUp->QueryInterface(IID_CATISpecObject,(void**)&myStartUpSpec);
```



Use the AddAttribute method from the CATISpecObject interface to define a new attribute



An attribute access mode is public by default, but can be set to private

- ***use the SetAccess method***

Attribute Definition Example

```
CATUnicodeString MyAttributeName = "Sketch";  
CATISpecAttribute* MySketchAttribute = NULL;  
MySketchAttribute = myStartUpSpec->AddAttribute( MyAttributeName,  
tk_specobject,  
sp_IN);
```

Definition of an input attribute
referring to another spec object

```
CATUnicodeString MyListAttributeName = "List";  
CATISpecAttribute* MyListAttribute = NULL;  
MyListAttribute = myStartUpSpec->AddAttribute( MyListAttributeName ,  
tk_list,tk_specobject,  
sp_IN);
```

Two ways to define an attribute
corresponding to a list of values

// Or

```
MyListAttribute = myStartUpSpec->AddAttribute( MyListAttributeName ,  
tk_list(tk_specobject),  
sp_IN);
```

Setting Attributes Default Values



To set attribute default value, use the appropriate method:

- **CATISpecAttribute::Setxxx**
 - *SetString* to assign a value to a *tk_string* type attribute,
 - *SetSpecObject*, *SetListString*, *SetDouble*, ...

```
CATISpecAttribute* MyDoubleAttribute = ...;  
MyDoubleAttribute->SetDouble(1.5);
```



To set an attribute value by using another attribute, use

- **CATISpecAttribute::SetSpecAttribute**

Feature Instantiation



To create an instance,

- **Open the Catalog in read only**
 - Use method `AccessCatalog` defined in file `CATCatalogFactoryServices.h`
- **Retrieve the Startup**
- **Instantiate the startup**
 - Use the `CATISpecObject::Instantiate` method
- **Set attributes values**



Open the catalog

```
CATContainer *piSpecCont = ...;  
CATUnicodeString catalogName = "MyCatalog.CATfct";  
CATUnicodeString clientID = "ID";  
CATCatalog *piCatalog = NULL;  
rc = ::AccessCatalog( &catalogName, &clientID, piSpecCont, &piCatalog );
```

Feature Instantiation

– *Retrieve the Startup if not already done*

```
CATBaseUnknown* myStartUp = NULL;  
CATUnicodeString myStartUpName = "MyStartUp";  
rc = piCatalog->RetrieveSU(    &myStartUp,  
                             &myStartUpName,  
                             "CATISpecObject");
```

– *Instantiate the Startup in the feature container:*

```
CATISpecObject* piSpecOnStartup = NULL;  
rc = myStartUp->QueryInterface( IID_CATISpecObject,  
                               (void**) &piSpecOnStartup );  
  
CATISpecObject* piSpecOnInstance = piSpecOnStartup ->Instanciate (myStartUpName, piSpecCont);
```

Assigning values to feature attributes

- *Attributes should be accessed through the CATISpecAttrAccess interface.*
 - *Store an attribute key to speed up access*

```
static CATISpecAttrKey * _attKey = NULL;
```

- *Use GetXXX and SetXXX methods to access to the value*

Retrieve the access interface

```
CATISpecAttrAccess *piAccess = NULL;  
HRESULT rc = myFeature->QueryInterface( IID_CATISpecAttrAccess,  
                                         (void **) &piAccess);
```

```
if (!_attKey) {  
    _attKey = piAccess->GetAttrKey("MyAttribute");  
}
```

If necessary, initialize the attribute key

```
double myValue = piAccess->GetDouble(_attKey);
```

Retrieve the attribute value using its key

Implement the Feature Behaviors



Interfaces are implemented through the Object Modeler extension mechanism



At least you will want to:

- *Provide a build mechanism for your feature*
- *Encapsulate in your own interface(s) the access to the feature's attributes*



You may also want to implement some DS interfaces to better integrate your feature in CATIA V5.

Feature Build



To allow re-computation, a feature must implement the CATIBuild interface.



CATIBuild Interface has only one method:

```
virtual HRESULT Build();
```

- ***It, retrieves the input attributes values,***
- ***calculates the modified output and neutral attributes values,***
- ***stores the modified attributes values.***

User Catalog Management

- ➔ ***Catalog files are dedicated to store feature definition (startup).***
- ➔ ***A Catalog may store any number of startup.***
- ➔ ***Once created a catalog should never be deleted***
 - ***In fact a catalog has a unique ID which is used to identify the startups it contains. When a catalog is deleted and recreated, existing instances of feature will still reference the old catalog ID.***
 - ***Instead upgrade it.***
- ➔ ***Catalogs implement the interface CATIICatalog.***
 - ***Use this interface to add startups to a catalog***

User Catalog Services



DS provides several methods to manage user catalogs, they are defined in file CATCatalogFactoryServices.h

- **CreateCatalog**
 - *This method creates a new catalog.*
- **UpgradeCatalog**
 - *This method open a catalog for modification.*
 - *Use this method when you want to modify an existing catalog.*
- **AccessCatalog**
 - *This method open a catalog in read only for feature instantiation*
- **SaveCatalog**
 - *Save a new or an upgraded catalog*



To protect the use of your catalog, those methods takes a client ID.

Data Compatibility Rules



To ensure a ascendant data compatibility between subsequent release of your application you must follow some rules:

- ***On a Startup:***

- *You can add or delete attributes.*
- *You can change an attribute initial value.*
- *You cannot modify an attribute type or quality (in / out / neutral).*
- *You can change it's name.*

- ***On a Catalog***

- *You can add new startups, as long as their name are unique in all catalogs.*
- *You cannot delete a startup or move it to another catalog.*
- *You cannot change the name of catalog or try to create it twice.*
- *You cannot modify an attribute type or quality (in, out or neutral).*

Accessing to DS Catalogs



The CATCatalogFactoryServices methods do not work on DS catalogs.



In the current version of CAA your are able to open a DS catalog for startup derivation.

- *Use the method CATCatalogManager::OpenCatalog followed by CreateSUInCatalog on your catalog to create a startup that derive from a DS startup.*



In the subsequent version, you will not be able to open a DS catalog directly.

- *Instead you will use a factory method, that do both the OpenCatalog and the CreateSUInCatalog.*
- *There will be one factory method for every feature open to derivation*

About Features Extensions

In this lesson you will learn the concept of Features extensions

Feature Extension

Feature Extension



The Specification Modeler provides a mechanism to extend an existing feature by adding new attributes on it, without inheritance.

- A feature extension has its own attributes (neutral).*
- Its definition is stored in a catalog (.CATfct).*
- A feature extension is linked with an application.*
- An feature extension is created on a feature at run-time.*
- It is stored in an applicative container and can be activated and deactivated on demand.*

Feature Extension



The feature extension mechanism allows data sharing without any documents corruption.

A feature extension will be active in an existing document only if the application and the extensions catalog are presents.

if not, the feature behaviors are not impacted

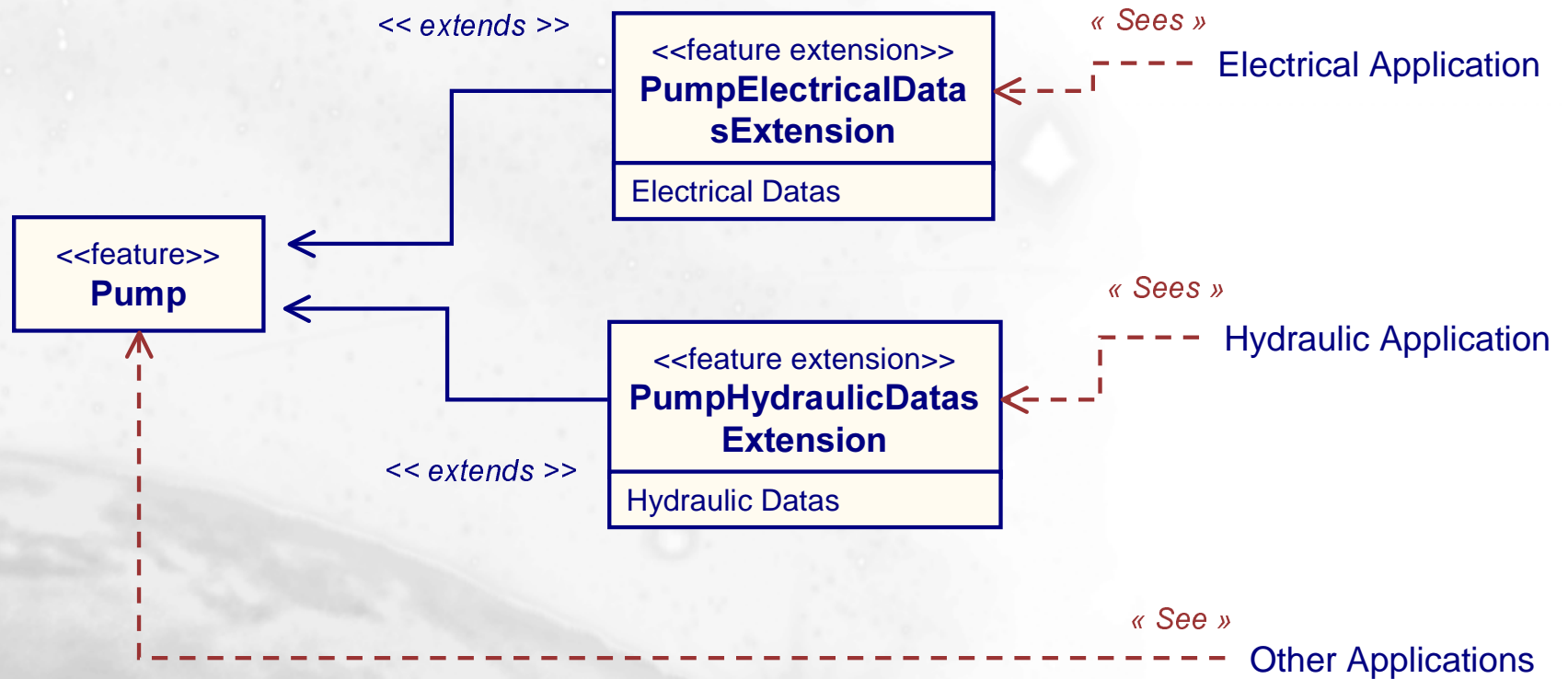
A feature extension has its own Interface to access these attributes.

Several feature extensions could extend a feature for one or different applications

Any feature extension could extends any feature.

Feature Extension

Example :



Programming Tasks

In this lesson you will learn how to implement a Feature Extension

- ▣ **Process to define a Feature Extension**
- ▣ **Define the feature extension data structure**
- ▣ **Create instance in document**

Process to define a Feature Extension



Define the feature extension data structure

- *Create the extension catalog*
- *Create the feature extension in the catalog*
- *Define the feature extension attributes*
- *Save the Catalog*



Implement on the feature extension the attributes access interface



Create instance in document

- *Create an applicative container for an applicative domain*
- *Open the extension catalog in the applicative container*
- *Activate the feature extension on the feature*

Process to define a Feature Extension



Management

- *Verify that a feature extension is active*
- *Access the feature extension's attributes*
- *Query all the active feature extensions of the feature*

Define the feature extension data structure

– *Create the extension catalog*

```
CATUnicodeString CatalogName = "MyExtCatalog.CATfct";  
CATICatalog* piMyExtCatalog = NULL;  
HRESULT rc = ::CreateCatalog( &CatalogName, &piMyExtCatalog);  
// Set a client identification for the catalog.  
CATUnicodeString ClientId = "MyClientId";  
rc = piMyExtCatalog -> SetClientId ( &ClientId );
```

– *Create the feature extension in the catalog*

```
CATIExtensionFactory *piFactoryOnMyExtCatalog = NULL;  
rc = piMyExtCatalog -> QueryInterface ( IID_CATIExtensionFactory, (void**) &piFactoryOnMyExtCatalog);  
const char *ExtensionLateType = "MyExtension";  
CATIExtension *piMyExt = NULL;  
rc = piFactoryOnMyExtCatalog -> CreateExtension ( ExtensionLateType ,  
                                                ExtensionSuperType, May be NULL if no inheritance  
                                                &piMyExt );
```

Define the feature extension data structure

– *Define the feature extension attributes*

```
CATISpecObject *piSpecOnMyExt = NULL;  
rc = piMyExt -> QueryInterface( IID_CATISpecObject, (void**) &piSpecOnMyExt );  
piMyExt -> Release( ); piMyExt = NULL;  
// example :  
CATUnicodeString AttributName = "...";  
// by default, the attribute's quality of a feature extension is NEUTRAL  
CATISpecAttribute *piAttribute = piSpecOnMyExt -> AddAttribute( AttributName, tk_string );  
piAttribute -> Release( ); piAttribute = NULL;
```

– *Save the catalog*

```
rc = ::SaveCatalog( &piMyExtCatalog, CatalogName );  
piMyExtCatalog -> Release(); piMyExtCatalog = NULL;
```

Create an instance in a document



Create an applicative container for an applicative domain

```
CATUnicodeString AppliContId = "ContainerName";  
CATUnicodeString ContainerLateType = "MyContainerType";  
CATUnicodeString ContainerSuperType = "CATFeatCont";  
CATBaseUnknown *pAppliCont = NULL;  
rc = ::CATCreateApplicativeContainer(& pAppliCont,  
                                     pDocument,  
                                     ContainerLateType ,  
                                     IID_CATIContainer,  
                                     ContainerSuperType,  
                                     AppliContId);
```

The container must be of type CATFeatCont
or must derive from a CATFeatCont

Pointer on the document (CATDocument *)

```
CATIContainer *piAppliCont = (CATIContainer*) pAppliCont;
```

Create an instance in a document



Open the extension catalog in the applicative container

```
CATCatalog *piMyExtCatalog = NULL;  
rc = ::AccessCatalog(&CatalogName, &ClientId, piAppliCont, piMyExtCatalog);  
piAppliCont -> Release();  
piAppliCont = NULL;  
piMyExtCatalog -> Release();  
piMyExtCatalog = NULL;  
  
// Get a CAT Extendable pointer on the feature to be extended  
CAT Extendable *piExtendableFeatureInstance = NULL;  
rc = FeatureInstance->QueryInterface( IID_CAT Extendable,  
                                     (void**) &piExtendableFeatureInstance );
```

Open the catalog inside the applicative container. From that point, extensions will be automatically instantiated in this applicative container.

Create an instance in a document



Activate the feature extension on the feature

```
CATBoolean CreatelfNecessary = TRUE;  
rc = piExtendableFeatureInstance -> ActivateExtension( ExtensionLateType ,  
                                                         AppliContId ,  
                                                         CreatelfNecessary );
```

The link between the feature (thru the CAT Extendable interface) and the feature extension is created at this step

About Providers

In this lesson you will learn the concept of Providers

- ▣ Providers
- ▣ Programming with providers

Providers



The Providers mechanism allows an external application using Feature extension to be integrated in the native behaviors of a DS application.



In other words, DS applications do not need to be aware of the specific contents of each applicative container or feature extension found in the document and yet still be capable of taking these into account when performing native tasks, such as navigation or visualization.



Providers are not persistent

Providers

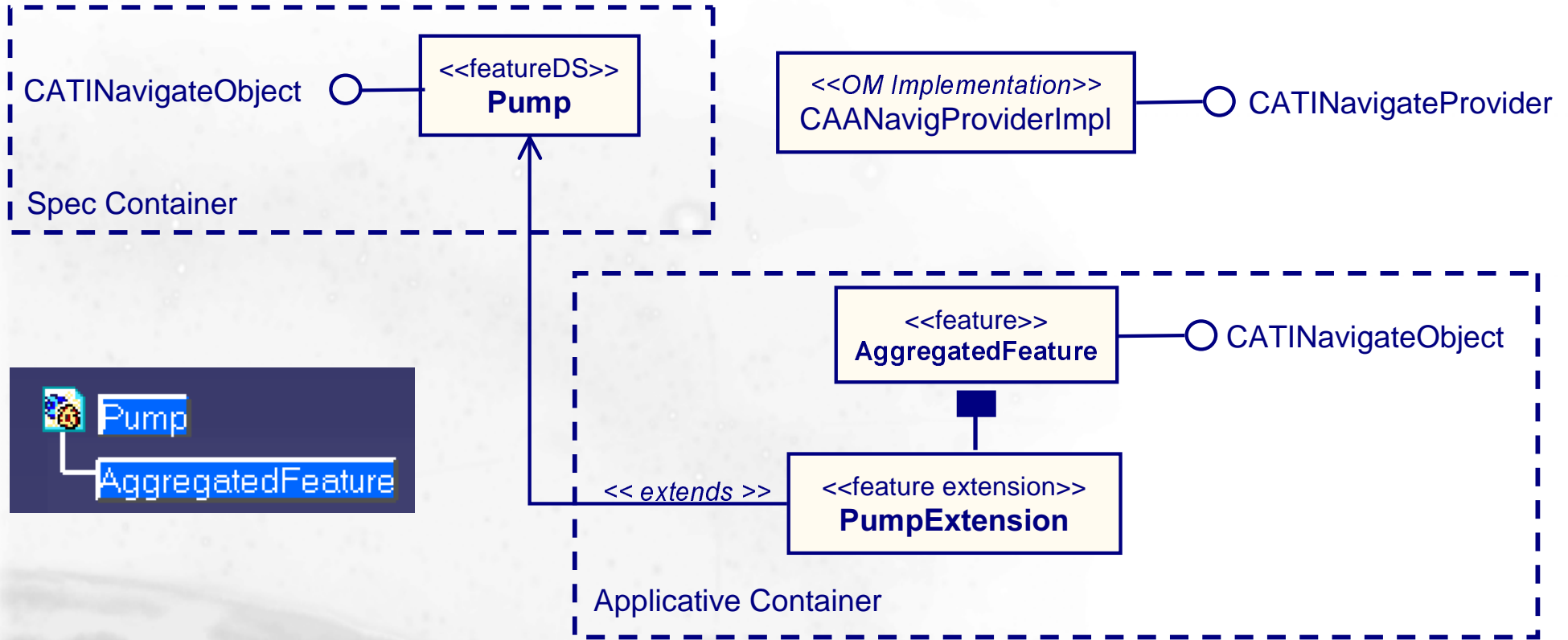


Five provider interfaces have been defined:

- *CATINavigateProvider* corresponding to *CATINavigateObject*.
- *CATI3DVisuProvider* corresponding to *CATI3DGeoVisu*.
- *CATIIconProvider* corresponding to *CATINavigModify*.
- *CATICtxMenuProvider* corresponding to *CATIUIActivate*.
- *CATIParmProvider* corresponding to *CATIParmPublisher*.

Providers

Example :



Using CATINavigateProvider allows to link the aggregated feature to the DS feature

Programming with Providers



Implement the provider interface

`<<OM Implementation>>
CAAProviderImpl`

—○ CATIxxxProvider



Declare Provider on the document after you create your applicative container

```
// Retrieve a CATIProviders pointer on the current document
CATIProviders *piProvidersOnDocument = NULL;
HRESULT rc = pDoc -> QueryInterface(IID_CATIProviders, (void**) &piProvidersOnDocument);

// Declare provider to list the children of the applicative container
CAAProviderImpl*pMyProvider = new CAAProviderImpl();
rc = piProvidersOnDocument -> AddProvider (CATIxxxProvider::ClassId(), pMyProvider);
```

Programming with Providers

Example : Implement CATINavigateProvider::GetChildren

```
HRESULT CAANavigProviderImpl::GetChildren(CATBaseUnknown * iObj ,
                                          CATLISTP(CATBaseUnknown) ** oListChildren)
{
    CATILinkableObject * piLinkableOnExtFeature = NULL;
    HRESULT rc = iObj -> QueryInterface (IID_CATILinkableObject, (void**) &piLinkableOnExtFeature);
    CATDocument *pDoc = NULL;
    if SUCCEEDED(rc) pDoc = piLinkableOnExtFeature -> GetDocument();
    else return E_FAIL;

    CATBaseUnknown * MyApplicativeContainer =NULL;
    CATIdent idAppliCont = "CATFeatCont";
    CATUnicodeString appliContName("MyApplicativeContainer");
    rc = ::CATGetApplicativeContainer ( &MyApplicativeContainer,
                                      pDoc,
                                      IID_CATIContainer,
                                      appliContName);
}
```

retrieve document

Retrieve the applicative container

Programming with Providers

Example : Implements CATINavigateProvider::GetChildren

```
CAAIFeatureExt *piFeatureExt = NULL;
rc = iObj -> QueryInterface (IID_ CAAIFeatureExt, (void**) & piFeatureExt);
if SUCCEEDED(rc)
{
    CATIExtendable * pIExtProd = NULL;
    rc = piFeatureExt ->QueryInterface(IID_ CATIExtendable, (void **) &pIExtendablePump);
    const char * ExtensionName = "PumpExtension";
    rc = pIExtendablePump->IsExtensionActive( ExtensionName,appliContName);
    if SUCCEEDED(rc)
    {
        CATIPumpExtension * pIPumpExt = NULL;
        rc = piFeatureExt ->QueryInterface(IID_ CATI PumpExtension, (void **) &pIExtendablePump);
        CATISpecObject * piSpec = pIExtendablePump->GetAggregatedFeature();

        (*oListChildren)->Append((CATBaseUnknown *)piSpec);
    }
}
return S_OK;
}
```

check if extension is activated

retrieve aggregated feature

Add aggregated feature to the list

To Sum Up ...

In this Course you have seen...

- **The V5 Feature concept**
- **The Feature creation and management**
- **How to implement a Feature**
- **How to extend a Feature**
- **How to integrate providers**

CAA V5 Visualization

You will learn how to manage object representation

- **Framework Objectives**
- **Model / View / Controller Architecture**
- **Visualization Interfaces**
 - *CATIVisu*
 - *CATIModelEvents*

Framework Objectives



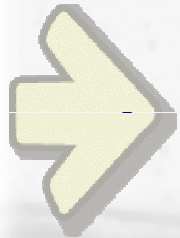
Visualize a model

- A model is visualized thanks to a graphic representation



Multi-window management

- Components can be seen in several viewers of the same or of several windows at the same time



Direct manipulation

- Viewers provide 3D and 2D manipulators to interact with the representation

Architecture Principles



Model / View / Controller

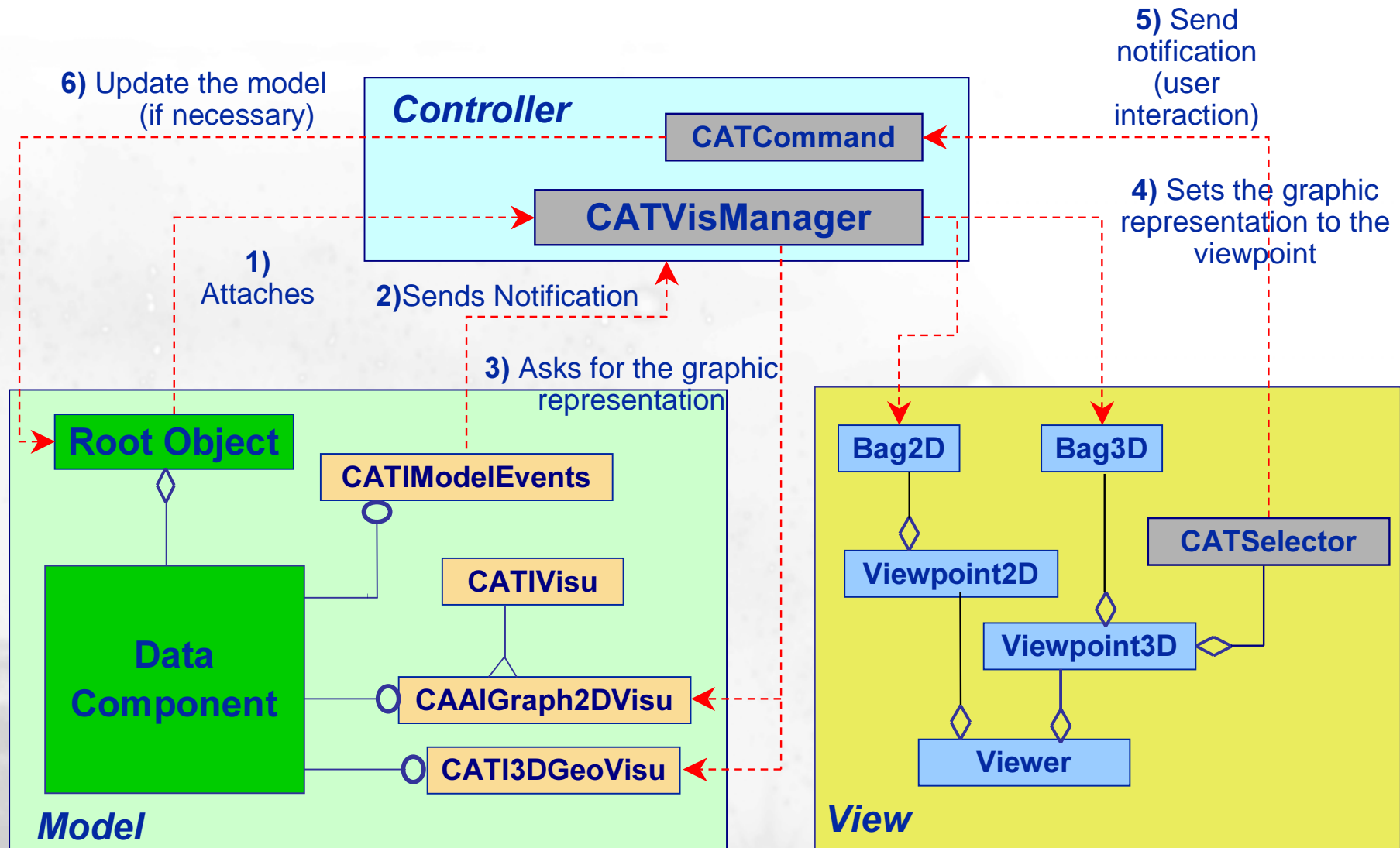
- Assume the independence between the data model and the presentation (view)
- Allow to change the viewer without impact on the data



Visualization is ruled by the controller

- Manage one or several views of the data
- Notify the views when the data change
- Receive the user interactions to update the model when necessary (object creation, modification or deletion)

Architecture Overview

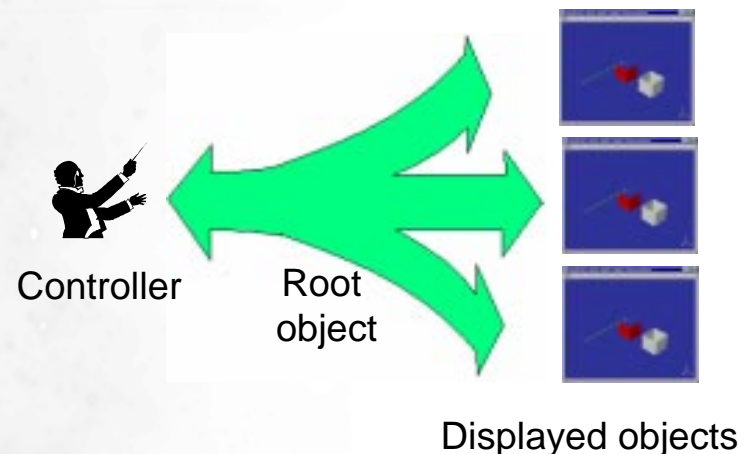


Controller Missions



The controller displays a root object (federative model entity) :

- **It gets representations from the root object**
 - **It displays the root object in every viewer and every viewpoint defined in the document window.**
- **The root object asks representations for its children to the controller**
 - **It groups them in a bag of representations**
 - **Children may also be federative entities (recursive process)**

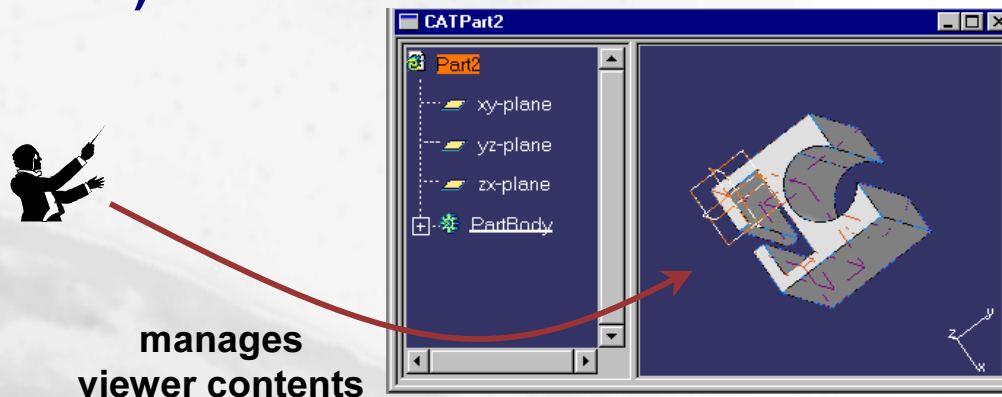


Controller Missions



The controller refreshes viewers according to model changes

- **Listens to events sent by the model**
 - **Creation** : Ask for new representations (if any)
 - **Modification** : Modify or re-computes the representation
 - **Deletion** : Removes obsolete representations (if required)



Controller Creation



The controller CATVisManager is unique

- You are not responsible to create or delete it**
- You can retrieve it by a global access method :**

```
CATVisManager::GetVisManager();
```

Root Object Attachment



Attachment from the Root Object to the Controller

- It creates one or several (i.e. 2D/3D) cells of visualization



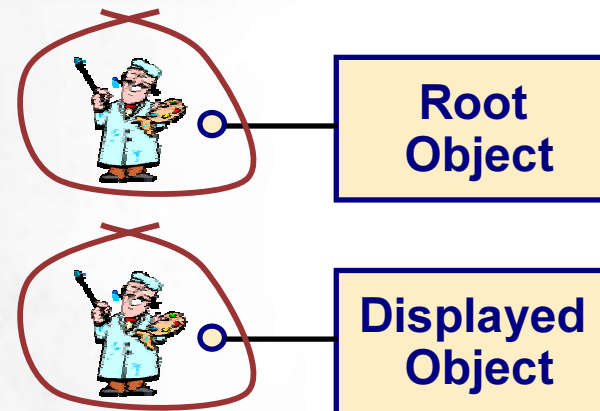
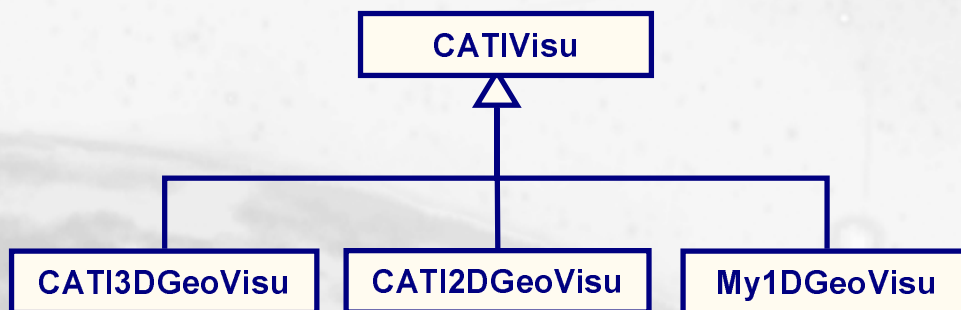
A cell of visualization is :

- A root object
- A viewpoint
- A list of visualization interfaces used by the controller
- A CATSelector that allows the user to interact with the representation

Controller Protocol (1/4)



- The controller has to know the Visualization interfaces
- The object behavior is described through a dedicated interface that derives from CATIVisu.
 - The root object and all its descendants adhere to it.



Controller Protocol (2/4)



Controller uses the Visualization interfaces to:

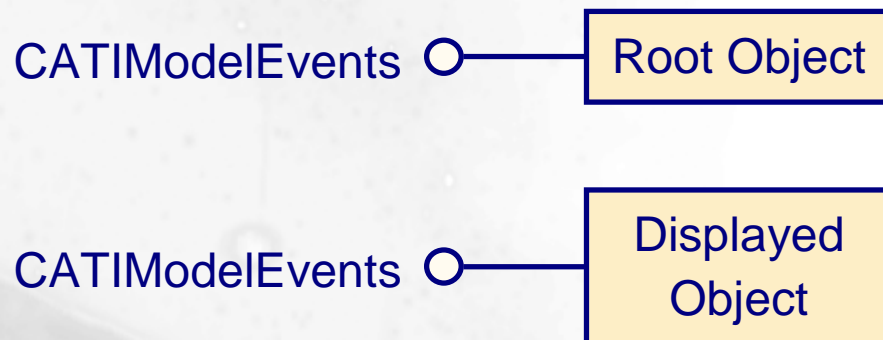
- Create the initial graphic representation**
 - BuildRep method**
- Update the representation**
 - ModifyRep method**
- Build the representation for (pre)highlight purposes**
 - BuildHighlightLook method**
- Build the list of selected elements when the representation is selected**
 - DecodeGraphic method**

Controller Protocol (3/4)



Connection from Model to Visualization world

- CATIModelEvents sends events when the model is updated
- These events inform the controller when re-computation is needed
- The root object and its descendants adhere to it



Controller Protocol (4/4)



User Interaction

- A CATSelector is created with a cell of visualization to answer to user actions (Select, Pick, Drag, ...)
- A CATCommand can be used to catch and process the notifications send by the CATSelector
- The CATCommand is associated to the CATSelector during the Root Object attachment

Events Tree

// When you aggregate the displayed object to the root

```
CATIModelEvents _var spModelEventsOnRoot(piRoot);  
piModelEventsOnRoot->ConnectTo((CATBaseUnknown*)DisplayedObject);
```





// When you want to update the visualisation

```
CATCreate Createnotif(DisplayedObject);
```

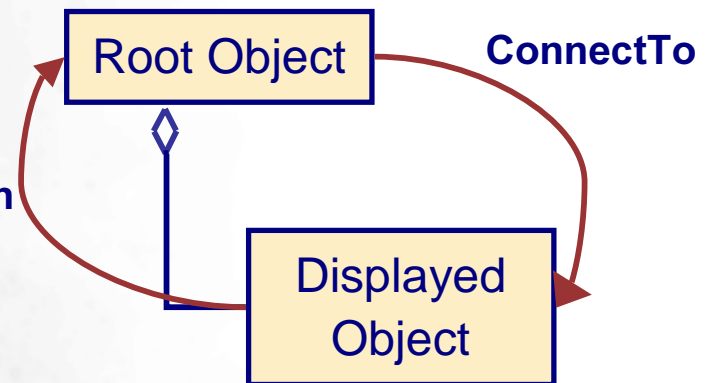
```
CATIModelEvents _var spModelEventsOnObject(DisplayedObject);  
spModelEventsOnObject->GetDispatcher(Cratenotif);
```



A notification can be

-  **CATCreate**
-  **CATModify**
-  **CATDelete**
-  **CATModifyVisProperties**

SendNotification



Visualization Interfaces



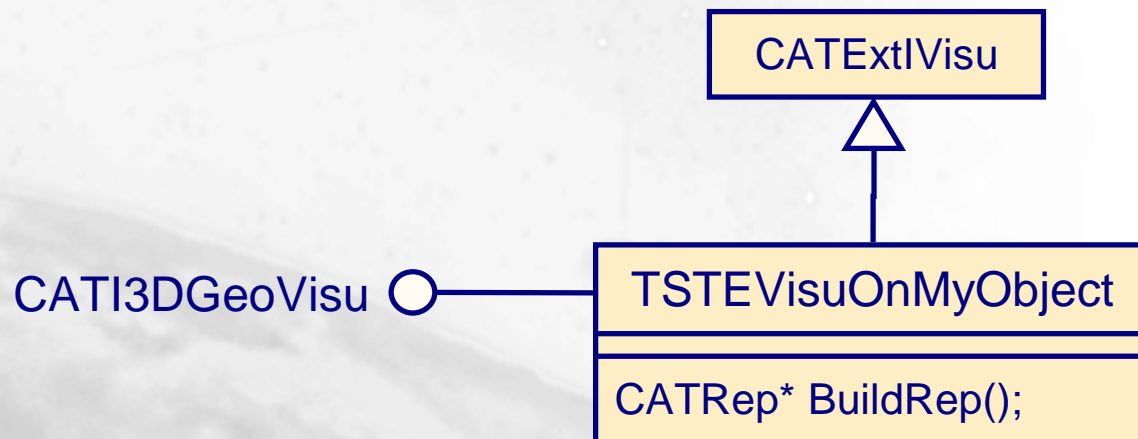
CATVisu Implementation

➔ Adapter class : CATExtlVisu

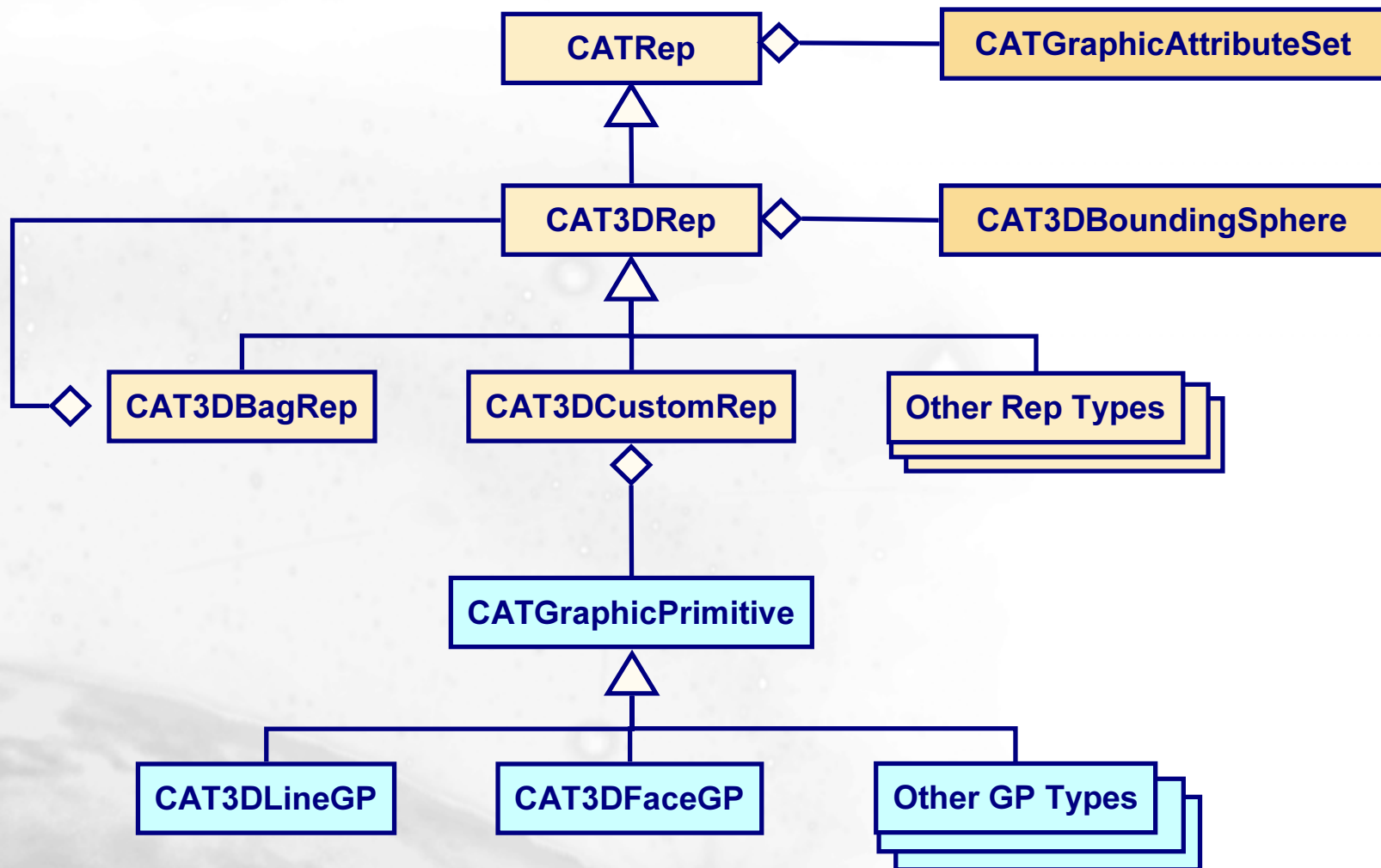
- It provides almost all requested functions except BuildRep

➔ 2 default interfaces : The same object can be represented in 2D and/or 3D

- CATI2DGeoVisu and/or CATI3DGeoVisu
- the BuildRep method returns a pointer to a CATRep.



Representations

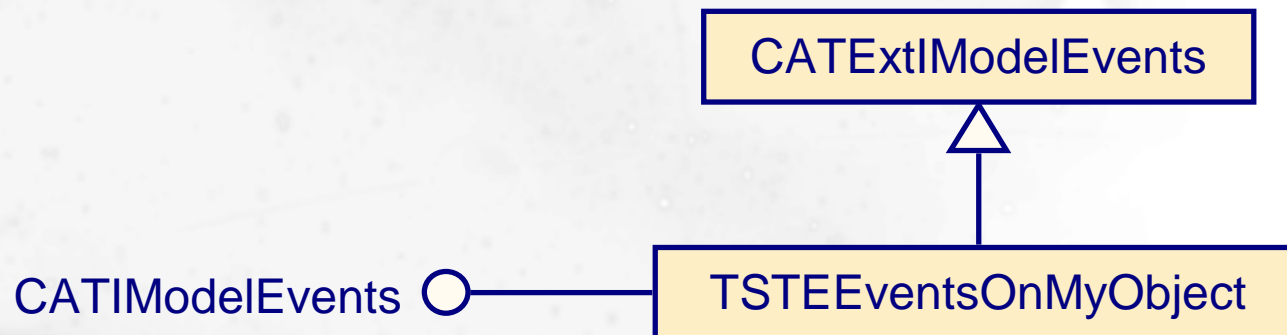


CATIModelEvents Implementation



Adapter class: CATExtIModelEvents

- No need to re-implement any method



To Sum Up ...

In this lesson you have seen...

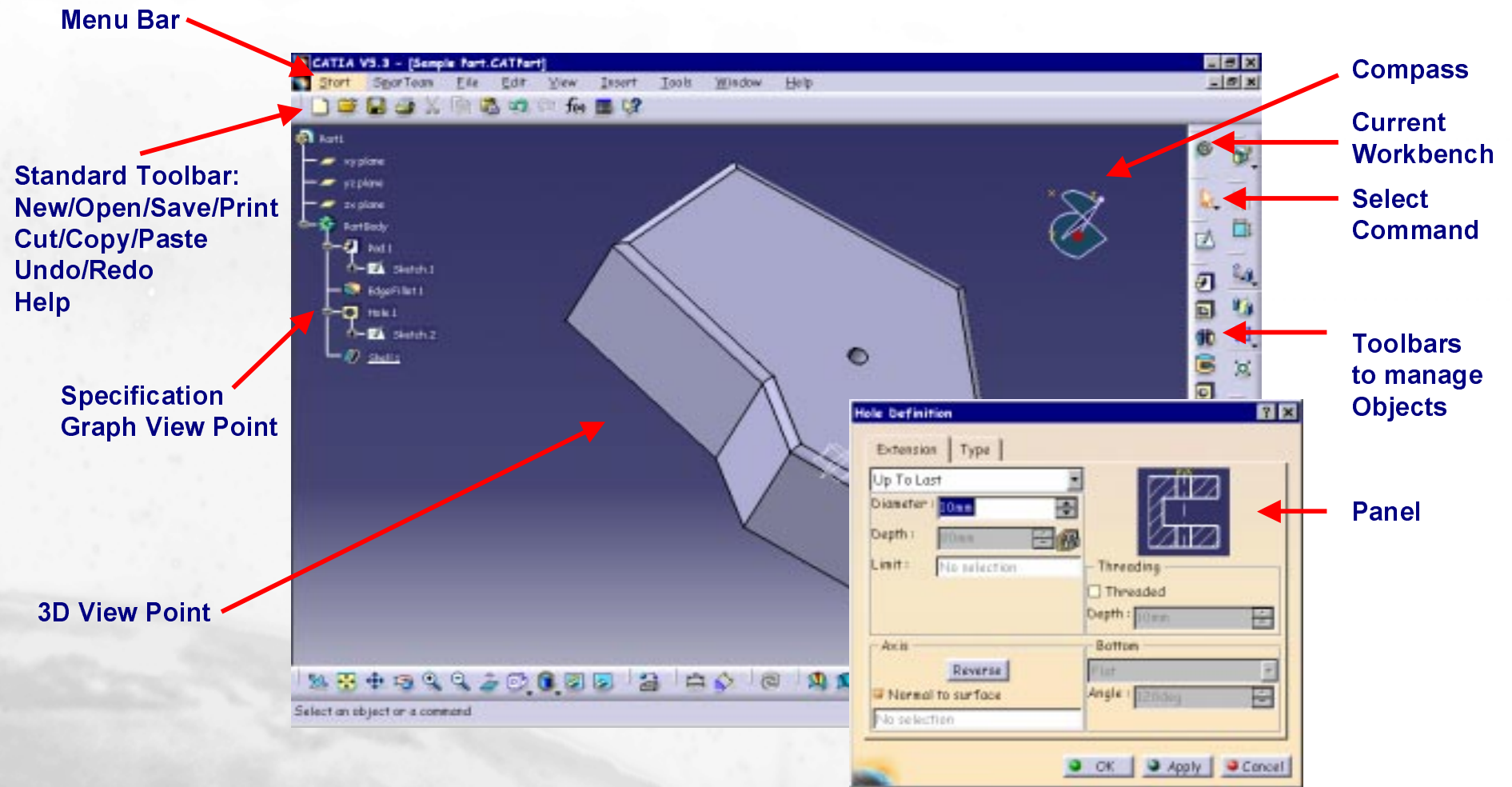
- **Model/View/Controller principle**
- **The interfaces which manage the Visualization of Features**

ApplicationFrame

In this lesson you will learn the CATIA V5 frame and Documents edition infrastructure

ApplicationFrame

Look and Feel of the CATIA V5 frame



Workshop and Workbench



- A workshop (or a workbench) defines a list of commands accessible through menus or toolbars.
- A workshop defines the common menus and toolbars associated to a given document type.
- A workbench provides specialized commands.
 - Part Design for solid modeling
- Several workbenches may be associated to a given workshop
 - For the CATPart document, several workbenches are available:
 - Part Design, Wireframe and Surface Design, Free Style, Generative Shape Design

A Model for Command Access

➔ ***A command encapsulates a basic user action:***

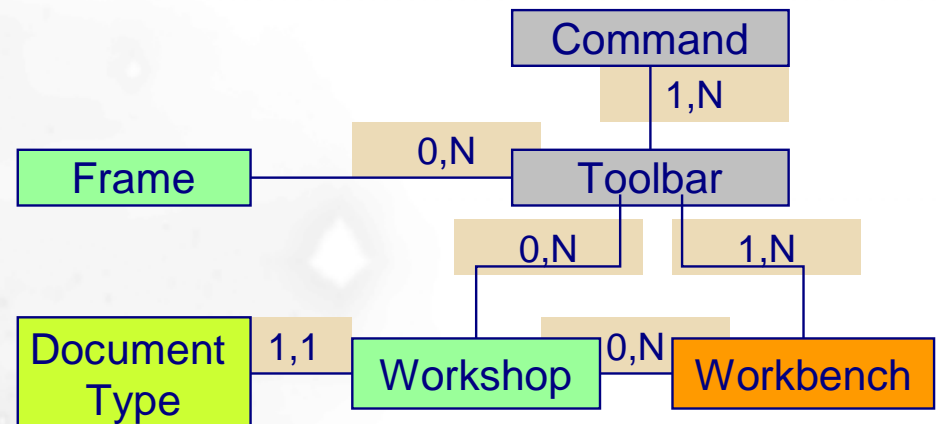
- *Creation, Edition, Analysis, Deletion*

➔ ***A workbench groups commands dedicated to a given domain:***

- *Part Design, Generative Drafting, Assembly Design, ...*

➔ ***A workshop groups common commands to a set of workbenches for a given document type***

➔ ***The frame provides the generic commands for all the workshops***



Command Header



Command header hold necessary information to load the command:

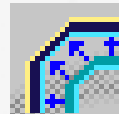
licensing management

availability conditions

pick ability

graphic appearance

normal, focused, pressed



text appearance

National Language Support

lazy-loading

load command code just-in-time

Multi-represented

Icon, Menu and Shortcut

Command Header Creation

MyWorkshop.cpp

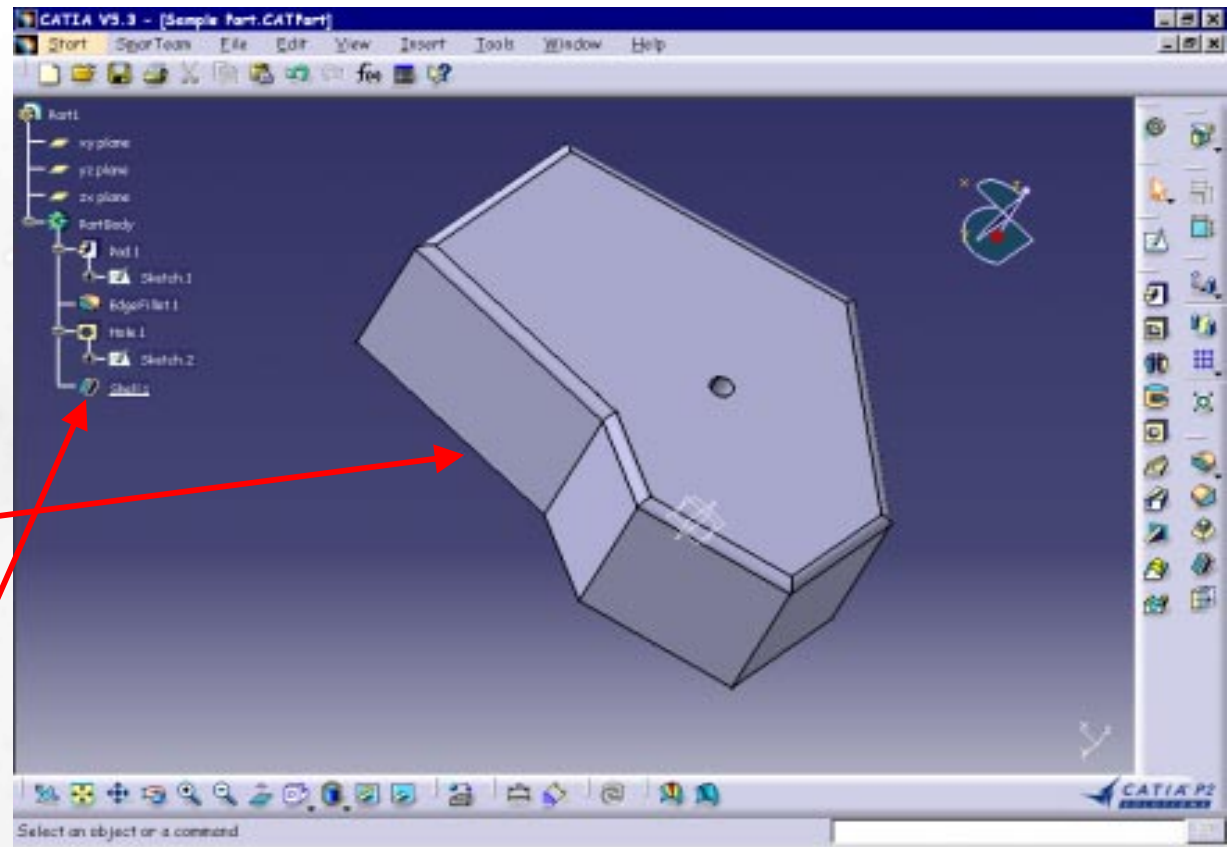
```
#include "CATCommandHeader.h"
MacDeclareHeader( CATMyCmdHeader )

void MyWorkshop::CreateCommands()
{
    new CATMyCmdHeader( "CommandName" ,
                        "SharedLibraryDefiningTheCommand" ,
                        "CommandClassName" ,
                        (void *) DataToBePassedToTheCommand );
    ...
}
```


Presentations

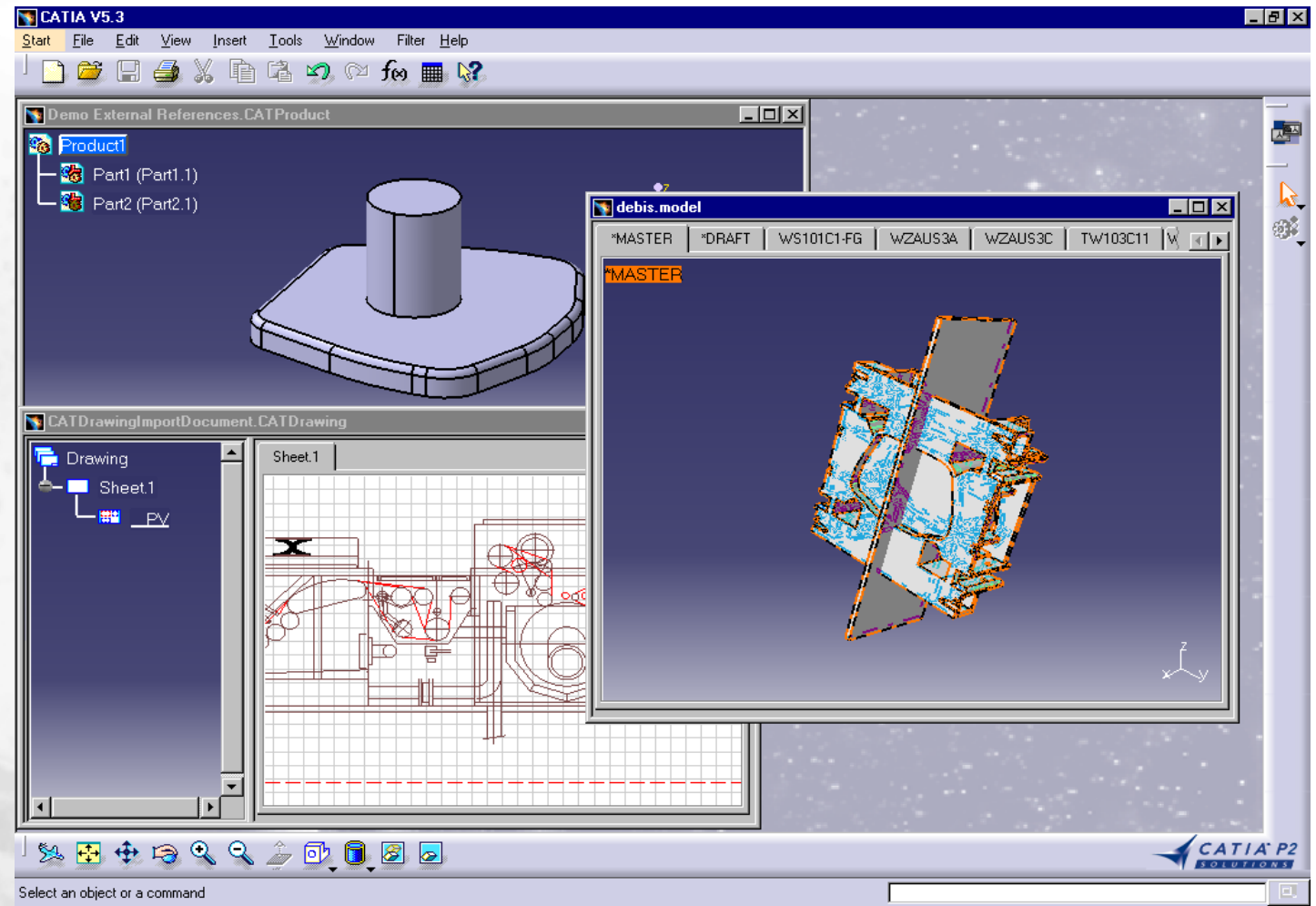
The document infrastructure defines the way it is displayed:

- a 2D view point
- a 3D view point
- a graph view point
- a combination

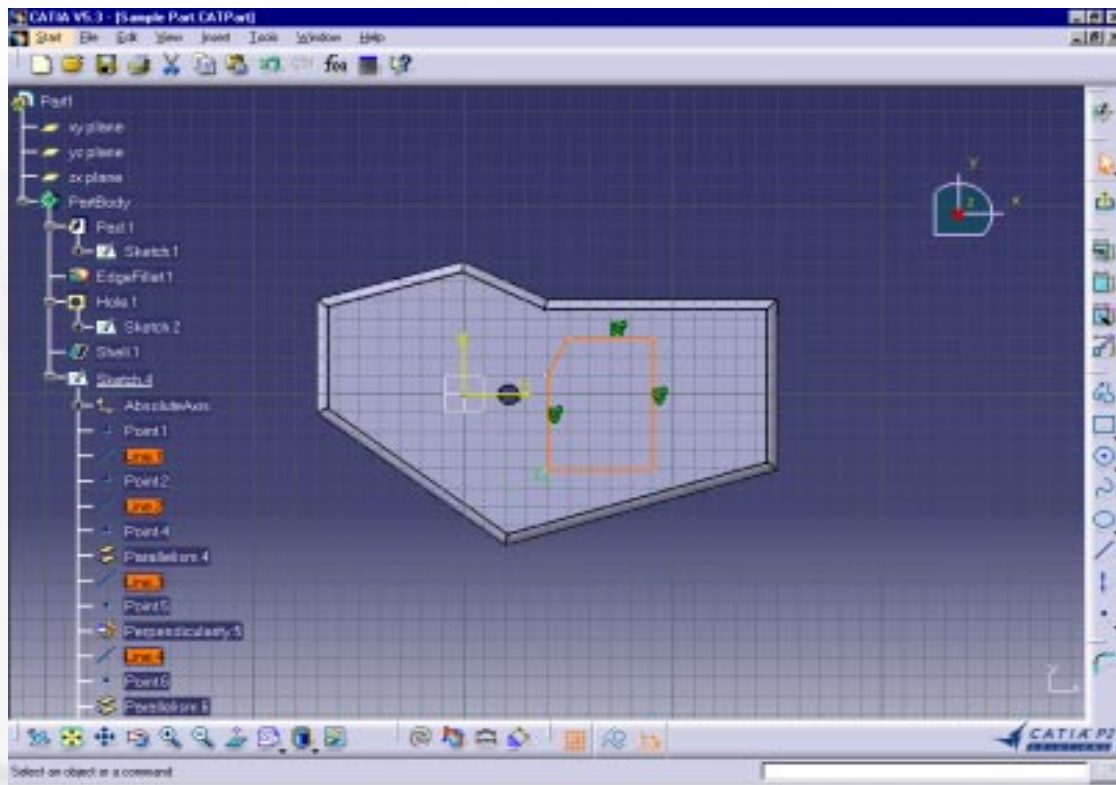


Presentations

- *Several documents can be displayed at the same time*
- **MDI: Multiple Document Infrastructure**



Path To Objects



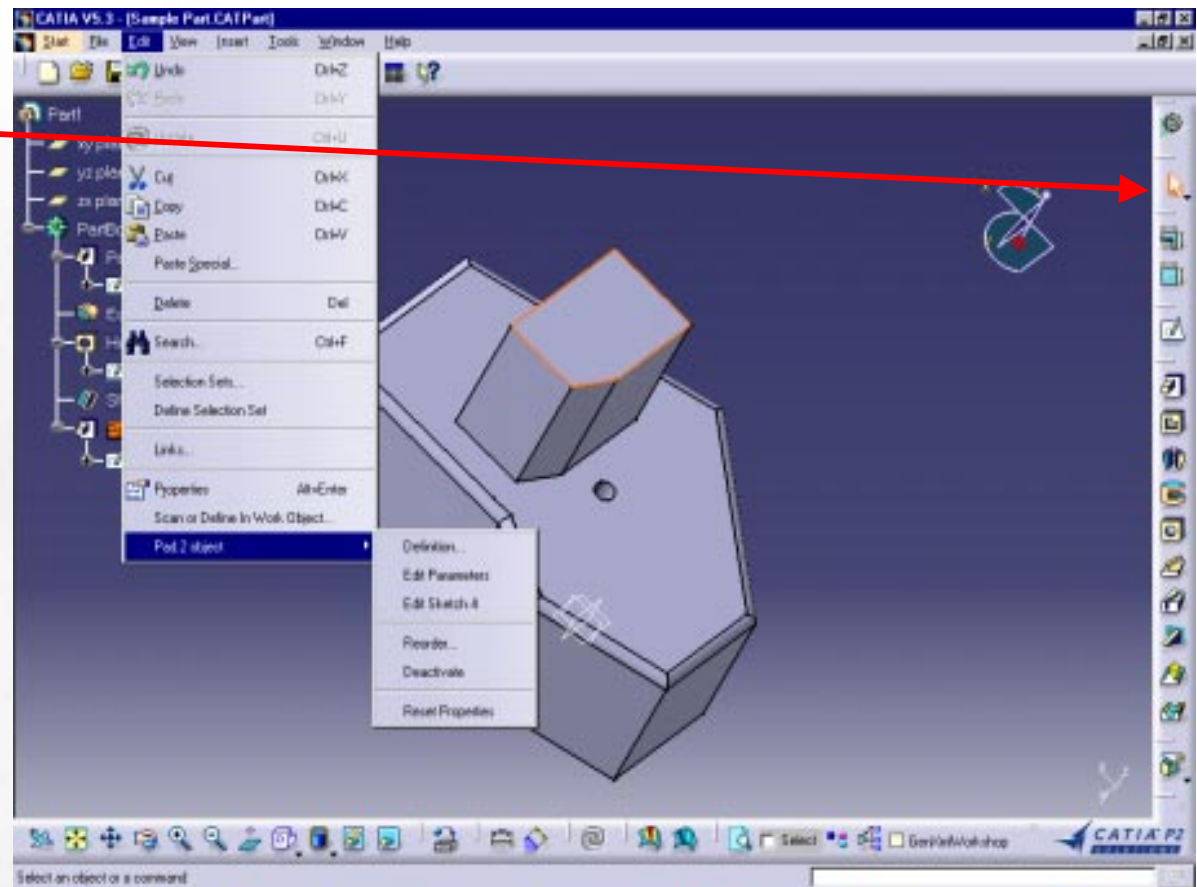
Access an object is done thru its ancestors.

EX: when selecting a line defined in a sketch, the complete path to access the line is retrieved through a CATPathElement= \Part1\PartBody\Sketch4\Line

The Select Command

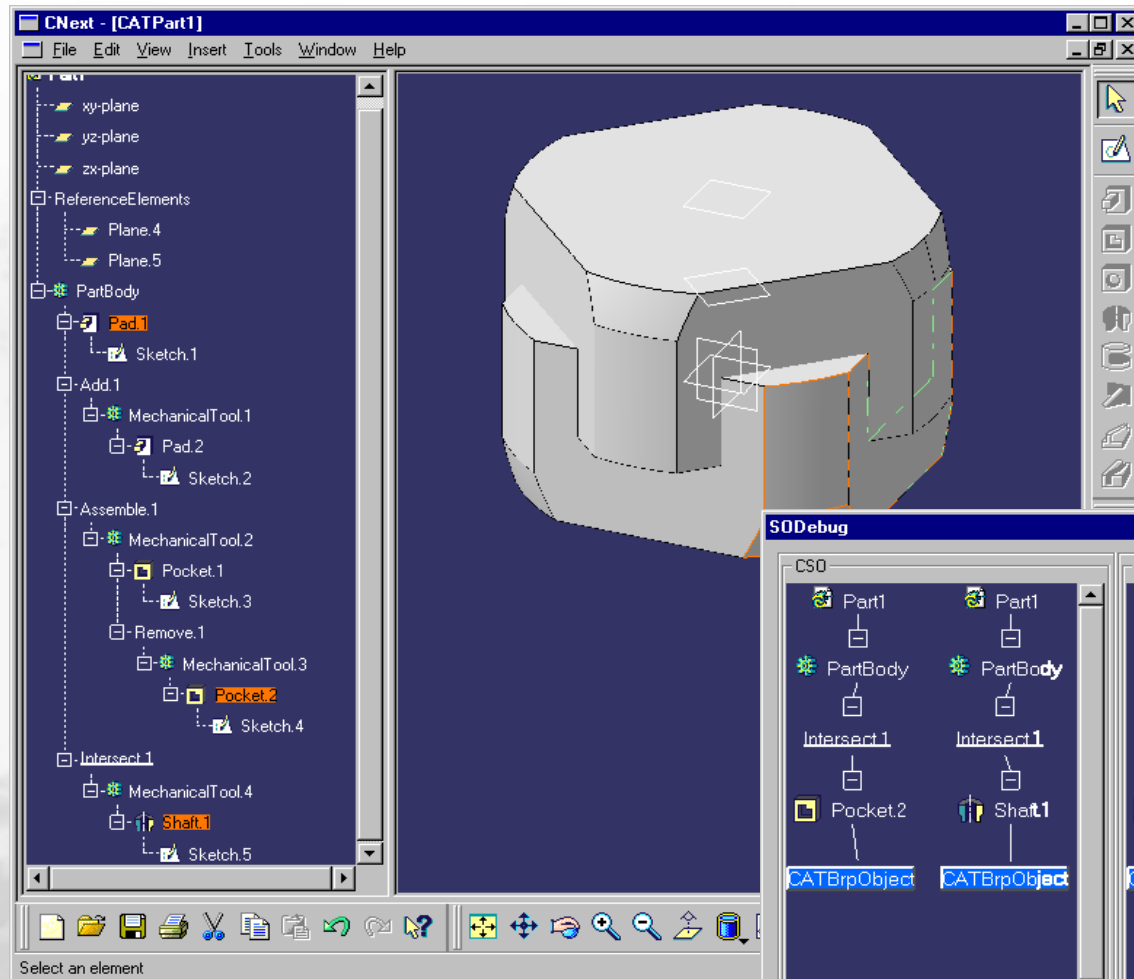
SELECT is the default command:

- ***Copy / Cut / Paste them***
- ***Delete them***
- ***Provide them as input for command to be launched (object/action paradigm)***

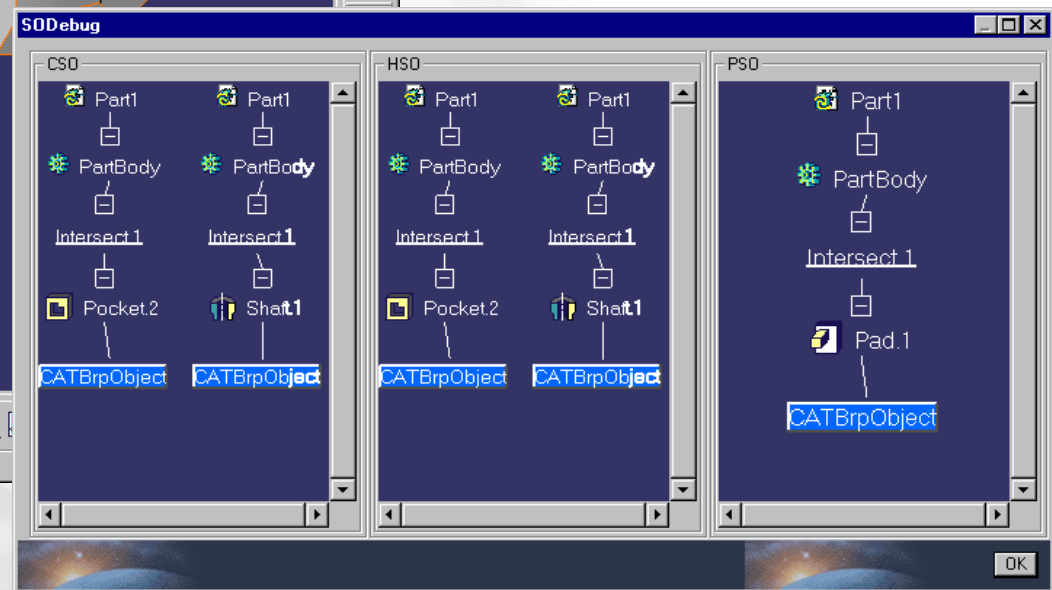


Selection offers also a mean to provide manipulators on objects.

Selection management



Each document is controlled by an editor managing the selection.



Selection management

Group of objects:



CSO: Current Set of Object

- *can be used by any command to retrieve the selected objects.*
- *Objects contained in the CSO have usually their graphical representation highlighted*



CATXSO: specialized by the following objects, known by the viewer:

- **HSO: Highlighted Set of Objects:** *the objects that the command highlights. Each object in the CSO is also in the HSO*
- **PSO: Pre-selected Set of Objects:** *the objects that are handled by a manipulator set by the current command, and that are pre-activated and moved.*



ISO: Interactive Set of Objects:

- *used to contain interactive objects, that is objects that are not part of the document, but which are displayed to enable their document object handling, such as manipulator handles.*

Object Edition

Object Edition is enabled in a way specified by the object:



if possible, an appropriate workshop is loaded

- depending on the fact that the **CATIUIActivate** interface is implemented by the active object (whose edition is requested), or by one of its ancestors
- workshops are not reserved to document types (example: Sketch)



additionally an edition command may be started

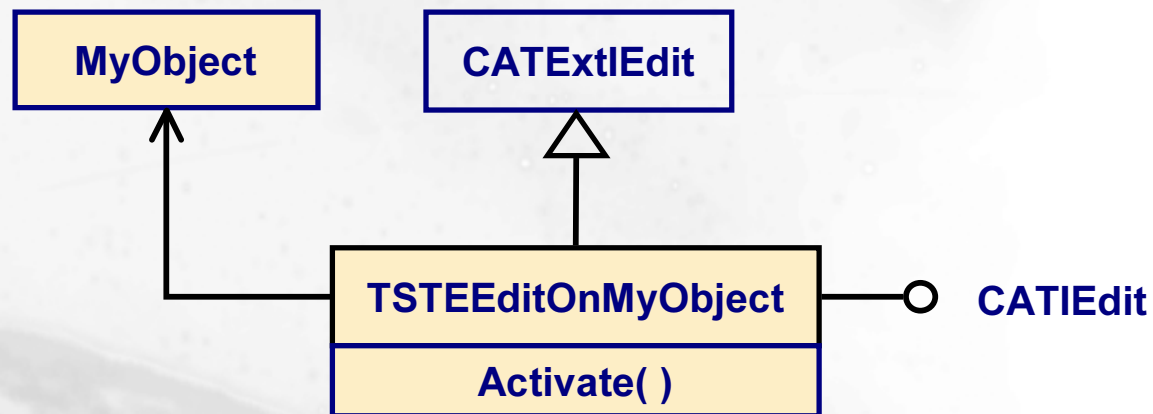
- depending on the fact that the **CATIEdit** interface is implemented by the active object (return the CATCommand class instance that enables the object edition)
- Activate method is called whenever the end user intends to edit the object by double clicking or through the object's contextual menu.

Object Edition

Implementing CATIEdit Interface on an object



create an extension class that derives from CATExtIEdit class and extends your object class (or late type) as a data extension. Then implement the Activate() method.



Activate makes new Command which is used for creation and modification

Object Properties



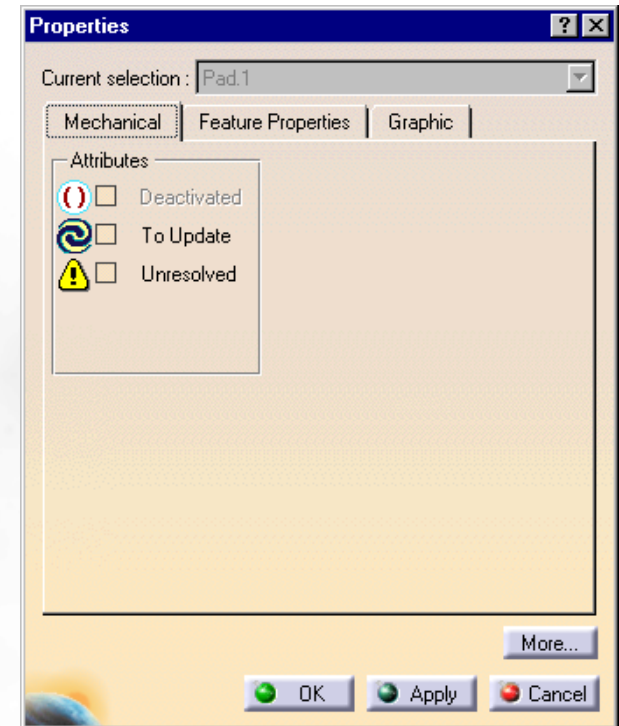
To add Properties for an object type:

- *If it's a DS object, use **CATIProperty***
- *If not, creates and implements your interface to define properties*
- *implement the **CATIEditProperties** interface to add your own tab page in both cases*



Object Properties edition can be triggered thru :

- *Selecting it, then 'Edit' menu bar*
- *Selecting it, then pressing **Alt+Enter***
- ***contextual menu item***



Application Properties

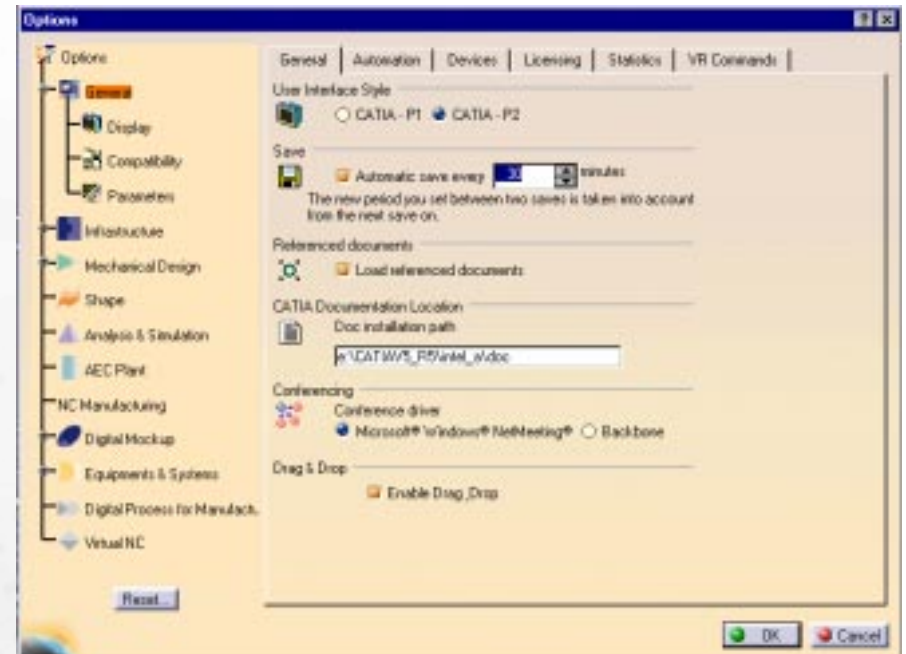
Application Properties are edited thru the Tools+Options command:

all global parameters are specified here, some are specific to an application, most are shared among all of them.

their values are kept in «settings» repositories sometimes both in administrator reference and in user preferences

- ***C:\Winnt\Profiles\User\Application Data\Dassault Systemes\CATSettings***

the interface to implement when adding a new tab page is CATIUserSettings



Application Integration



Two ways to integrate your commands in the CATIA V5 frame:

- Define a Addin in a specific context:***
 - add some toolbars in an existing workbench*
- Define a new workbench linked to a specific workshop***

Addin

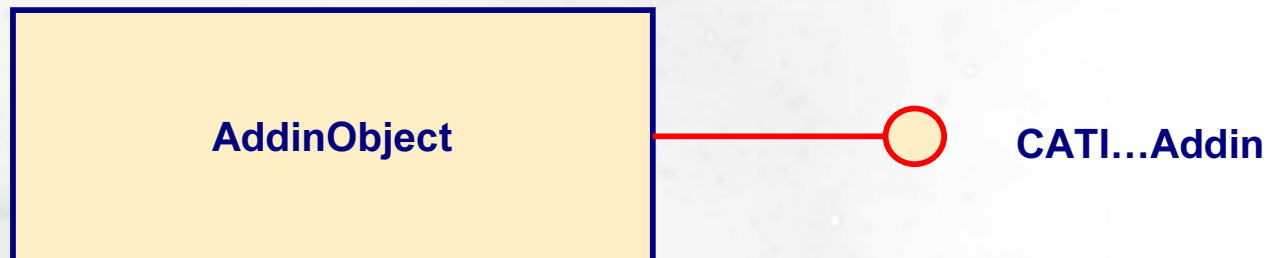


Define an object that implements the Addin interface of an existing workbench (Toolbar but not menu)

Two methods to implement:

void AddinObject::CreateCommands()

CATCmdContainer * AddinObject::CreateToolbars()



Workbench



Define an object that implements the Workbench interface of an existing workshop (Toolbar and menu)



Define an Addin interface for your new workbench



Two methods to implement:

void WorkbenchObject::CreateCommands()

CATCmdWorkbench * WorkbenchObject ::CreateWorkbench()

Workshop / Workbench definition

Workshop.cpp

```
...  
CATCmdWorkbench * MyWorkbench::CreateWorkbench()  
{  
    NewAccess(CATCmdWorkbench,pMyWorkbench,NewWorkbench);  
  
    NewAccess(CATCmdContainer,pMyContainer,NewToolBar);  
    SetAccessCustomerName(pMyContainer,"MyToolbar");  
  
    NewAccess(CATCmdStarter,pStarter1,NewButton1);  
    SetAccessCommand(pStarter1,"MyCommand1");  
    SetAccessChild(pMyContainer,pStarter1);  
  
    NewAccess(CATCmdStarter,pStarter2,NewButton2);  
    SetAccessCommand(pStarter2,"MyCommand2");  
    SetAccessNext(pStarter1,pStarter2);  
  
    AddToolBarView(pMyContainer,1,Right);  
    SetAccessChild(pMyWorkbench,pMyContainer);  
...  
}
```

Define the workbench

Define the container in which the commands will be instantiated. It can be a toolbar or a menu

Define a starter that will be associated to a command header and then arrange the command order

Define the container as a toolbar, position it by default on the right and include it in the workbench

Workshop / Workbench definition

Workshop.cpp

```
...  
CATCmdWorkbench * MyWorkbench::CreateWorkbench()  
{  
...  
    NewAccess(CATCmdContainer,pINSERT,INSERT);  
    NewAccess(CATCmdContainer,pMyMenu, MyMenu);  
    SetAccessChild(pMyMenu,pINSERT);  
  
    NewAccess(CATCmdStarter,pMyCommand1,MyCommand1);  
    SetAccessChild(pINSERT, pMyCommand1);  
    SetAccessCommand(pMyCommand1,"MyCommand");  
  
    SetWorkbenchMenu(CATCmdWorkbench1, pMyMenu);  
  
    return CATCmdWorkbench1;  
}
```

Retrieve the Insert Menu

Define another container to
be aggregated in the Insert
menu

Define the container as a
menu in the workbench

Main Interfaces to be implemented by an Object

CATICutAndPastable

Cut / Copy / Paste

CATIEditProperties

Define specific properties

CATIEdit

Edit

CATI2DGeoVisu

CATI3DGeoVisu

Display in 2D / 3D

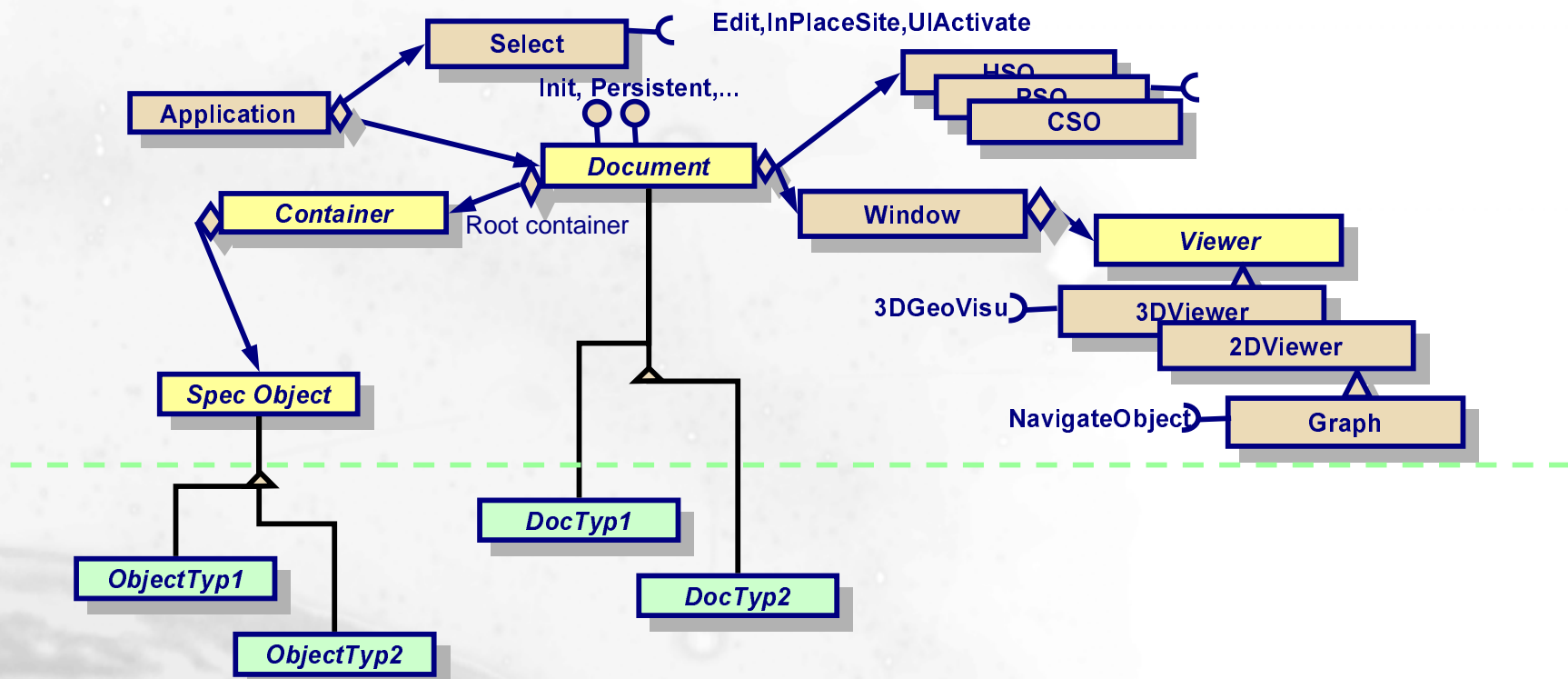
LifeCycleObject

Delete

CATINavigateObject

Display in the graph viewer

Additional Information : MVC model integration



To Sum Up ...

In this Course you have seen...

- **How to create Workbench and Toolbar**
- **How to integrate your commands (application)**
- **The selection of an object in the CATIA V5 frame**

CAA V5 DialogEngine

In this course you will learn the mechanisms necessary to describe and to manage the dialog of an interactive command, how to monitor the interactions at run time and manage Undo/Redo capabilities

- Objectives of CAA V5 DialogEngine
- Main notions
- How to define a new interactive command

CAA V5 DialogEngine Objectives



Provides all the objects and mechanisms necessary to describe and to manage the dialog of an interactive command



Monitors the interactions at run time



Manages Undo/Redo capabilities

Main notions

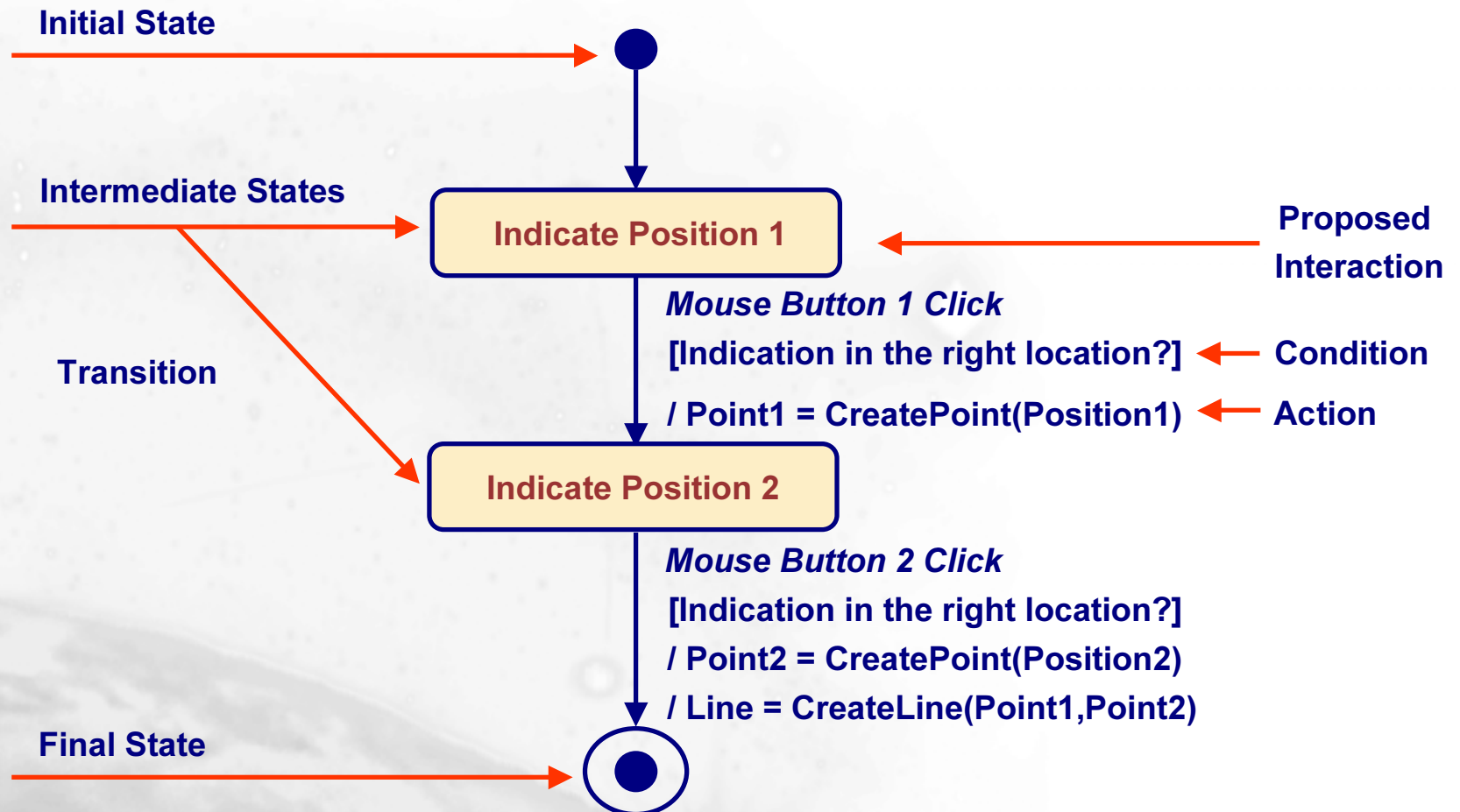
In this lesson you will learn the Commands management in CATIA V5

- ▣ **Notions to describe a dialog**
- ▣ **Finite State Machine**
- ▣ **Dialog Agent**
- ▣ **Dialog Command Life Cycle**
- ▣ **Interruption by another command**
- ▣ **Managing Undo/Redo**

Notions to describe a dialog

- A **State** is a step in a dialog where the program is waiting for an input.
- The type of the input is defined at each state.
- A **Transition** is defined between a source state and a target state.
- It is triggered by a **Condition**:
 - at least event-driven: an end user interaction
 - conditional, it validates the user input
- When a **Transition** is triggered, the target state becomes the active state.
- An **Action** can be executed during a transition.

Finite State Machine



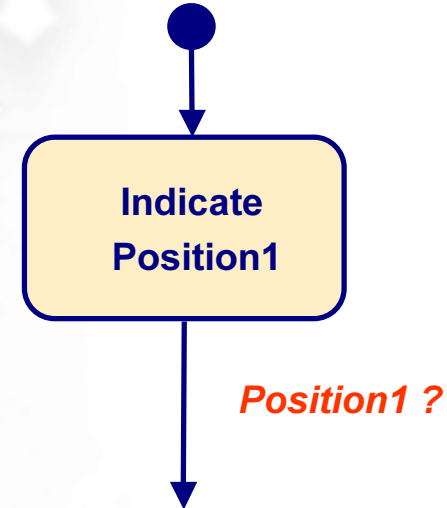
Dialog Agent

A Dialog Agent translates a user interaction into a user input

Ex: a CATIndicationAgent converts a left button mouse click in a 2D viewer as a 2D-coordinate input

It hides the details of how a user interaction is converted as a user input.

It helps us define an **input-driven** dialog instead of an **event-driven** one.



Dialog Agent



It simplifies the State Charts when composite inputs are necessary at one state.



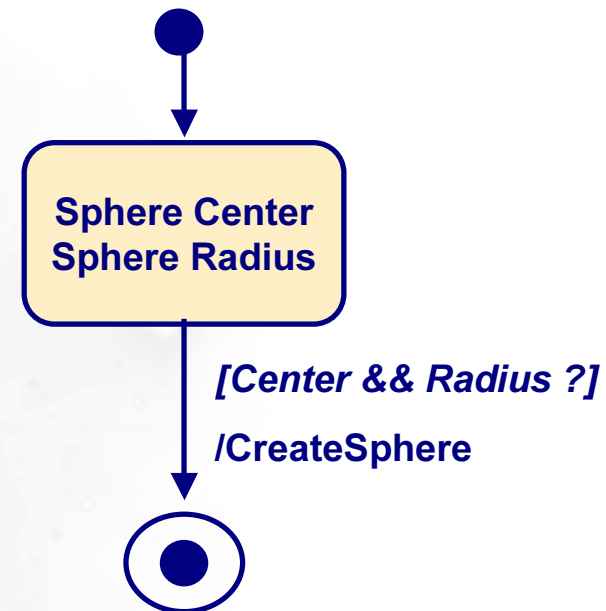
As soon as a dialog agent is valuated, it's not proposed anymore.

- *If this behavior is not suitable, the dialog agent needs to be defined as a repeater.*
- *SetBehavior(CATDIgEngRepeat)*

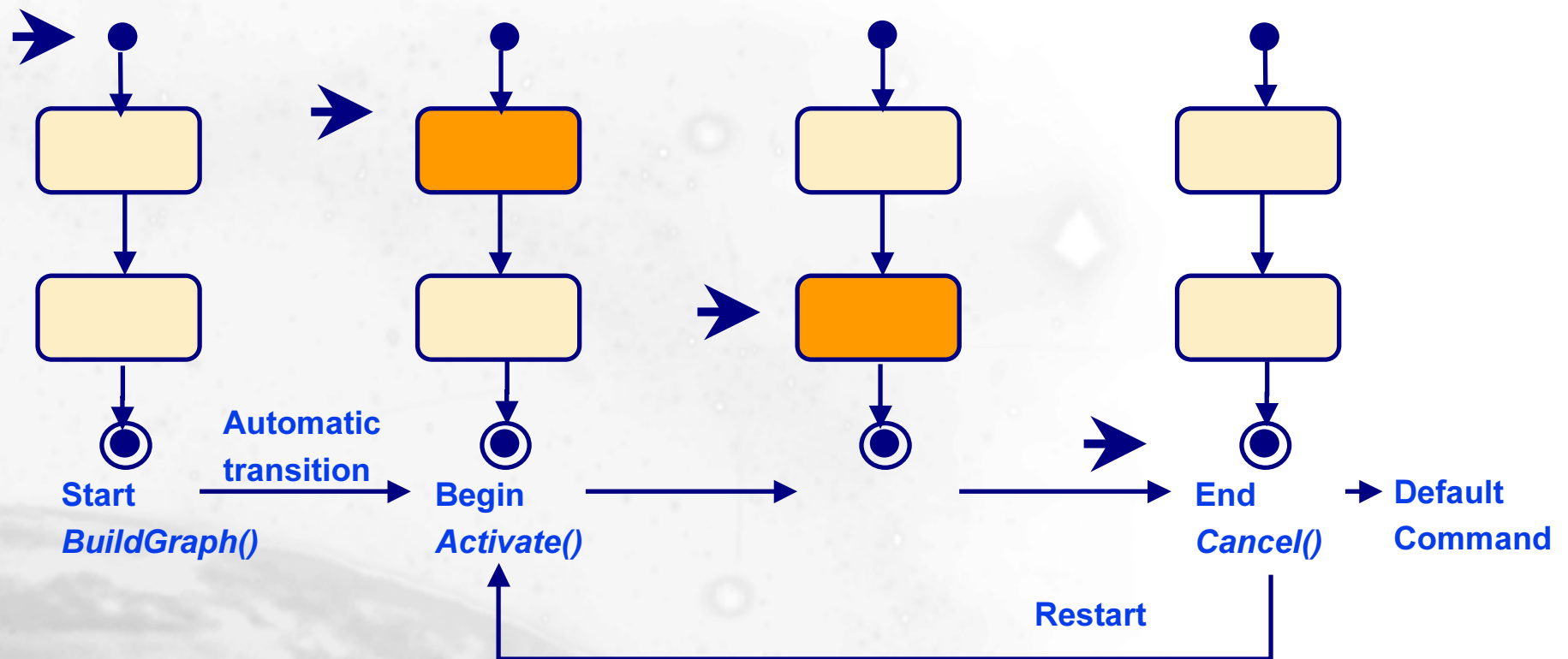


A dialog agent can be reused by recycling it after retrieving the input.

- *InitializeAcquisition()*



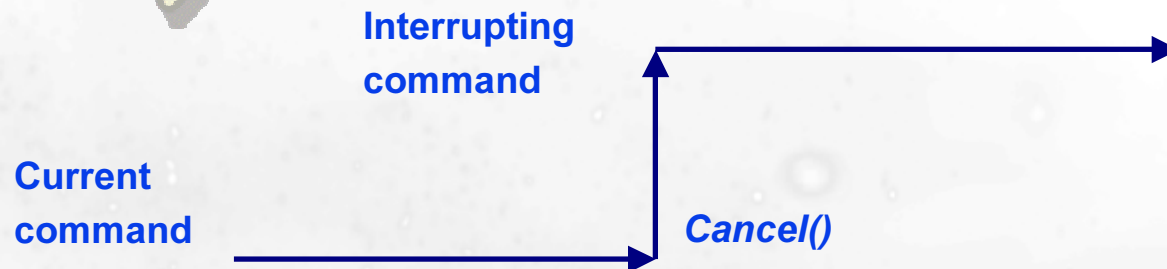
Dialog Command Life Cycle



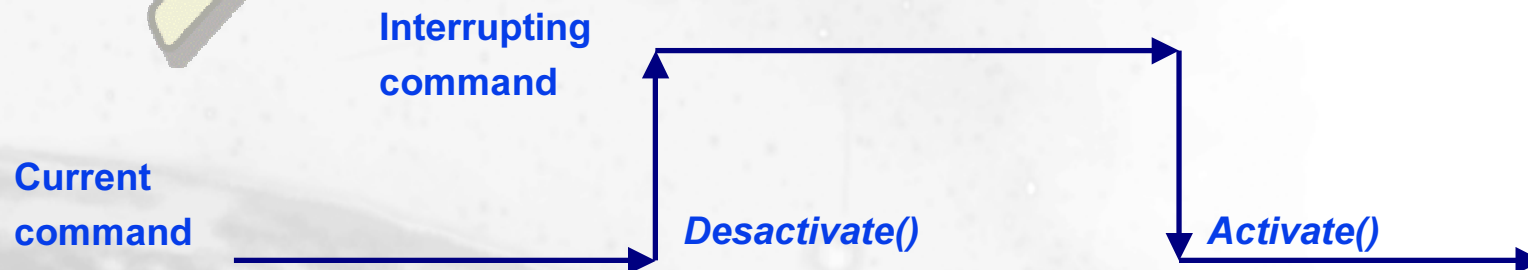
Interruption by another command



*The interrupting command is **exclusive***



*The interrupting command is **shared***



Managing Undo/Redo



Two levels of Undo/Redo

Input Undo/Redo within a command

Command Undo/Redo



When an action is associated to a transition, an Undo method should be defined

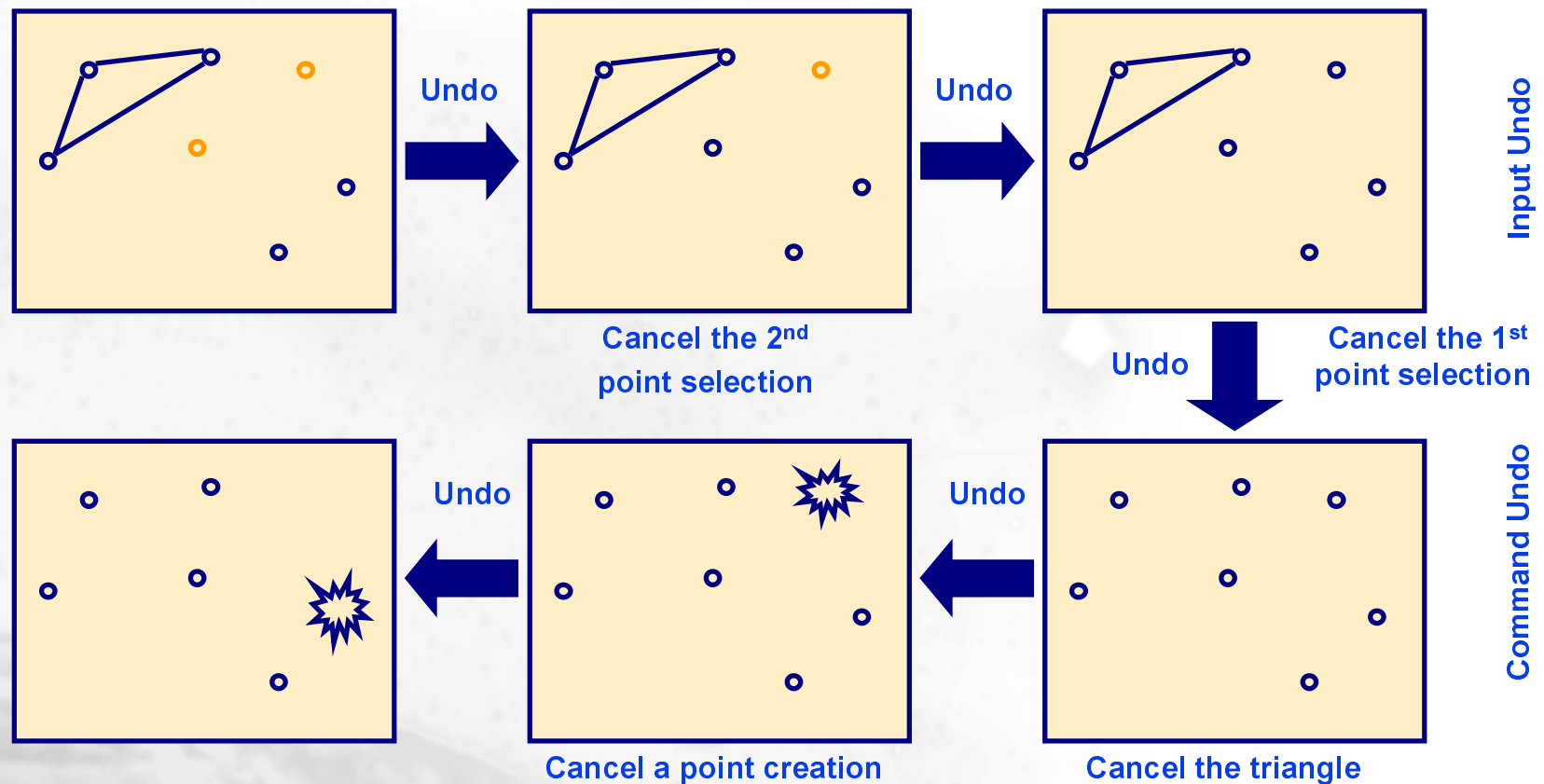


Within a command if the Undo command is activated, the previous transition is inverted and if there was an action, the corresponding Undo method is run.



Then, when out of a command, global Undo methods are executed to cancel the results of the previous commands

Managing Undo/Redo



How to program an interactive command

In this lesson you will learn how to program a new interactive command

- ▣ CATStateCommand
- ▣ Resource file
- ▣ Command with/without an argument
- ▣ Define the State Charts
- ▣ Define the States
- ▣ Dialog Agent Main Types
- ▣ Dialog Agent Behaviors
- ▣ Propose Multi Acquisition
- ▣ Retrieve Multi Acquisition
- ▣ Plug the Dialog Agents to the states
- ▣ Define simple condition to trigger transitions
- ▣ Define condition constraining the user input
- ▣ Define action done in a transition
- ▣ Rubber Banding

The CATStateCommand Class



- *Create a new class deriving from CATStateCommand*
- *Overload at least:*
 - *To describe your own state chart*
 - *BuildGraph()*
 - *To manage properly your command life cycle*
 - *Activate()*
 - *Desactivate()*
 - *Cancel()*
- *Define some specific methods that will be used as conditions or actions*
- *Store as data members the dialog agents used in the command*

Resource file

- *All the text messages that appear on the screen should be defined through a resource defined in a Resource file linked to the command.*

```
Class MyCommand: public CATStateCommand
```

```
{
```

```
    DeclareResource( MyCommand, CATStateCommand )
```

```
    ...
```

This defines a resource file MyCommand.CATNIs where all the command resources will be defined

```
MyCommand.myResource.Message="Select the first Point";
```

- *Resource files are stored in:*
 - *Catia\Code\intel_a\CNext\resources\msgcatalog*
 - *The English and default version is stored in this directory. The translated version is stored in a sub-directory linked to the language name:*
 - *French / German / Japanese*

Command with/without an argument

```
CATCreateClass( MyCommand )
```

```
...
```

This macro is mandatory to be able to instantiate a corresponding command header in a toolbar or a menu

- ***Sometimes it's interesting to pass an argument to a command, so the command can be reused for creation and also for edition***

```
CATCreateClassArg( MyCommand,CATISample )
```

```
MyCommand::MyCommand (CATISample * iArg)
```

```
...
```

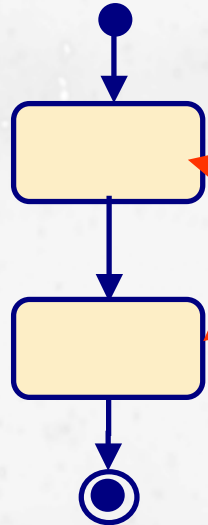
Define the State Charts



Overload the BuildGraph method

- ***Create all the states***
- ***Define the dialog agents you need***
- ***Plug the dialog agents in the corresponding states***
- ***Define the transitions between states:***
 - ***Source and target states***
 - ***Condition***
 - ***Action***

Define the States



```
void MyCommand::BuildGraph()
```

```
{
```

The initial state is already created.

```
    CATDialogState * State1 = GetInitialState("Start");
```

```
    CATDialogState * State2 = AddDialogState ("Second");
```

```
    ...
```

```
}
```

Dialog Agent Main Types

- *CATDialogAgent is the main class*
 - *This class can be used to define an agent linked to a panel object*

```
_AgentPanelOK = new CATDialogAgent(" AgentPanelOK ");  
_AgentPanelOK->AcceptOnNotify(_MyPanel,  
                               _MyPanel->GetDiaOKNotification());
```

- *CATIndicationAgent retrieves the coordinates of a 2D Point when clicking in a viewer.*
- *CATPathElementAgent retrieves the path element of the object under the mouse when clicking*

Dialog Agent Behaviors

...

```
_myAgent = new CATPathElementAgent( "myAgent" );
```

```
_myAgent ->AddElementType (CATISample::ClassName());
```

```
_myAgent ->SetBehavior(CATDIgEngWithPSOHSO|CATDIgEngWithPrevaluation);
```

```
AddCSOClient( _ myAgent );
```

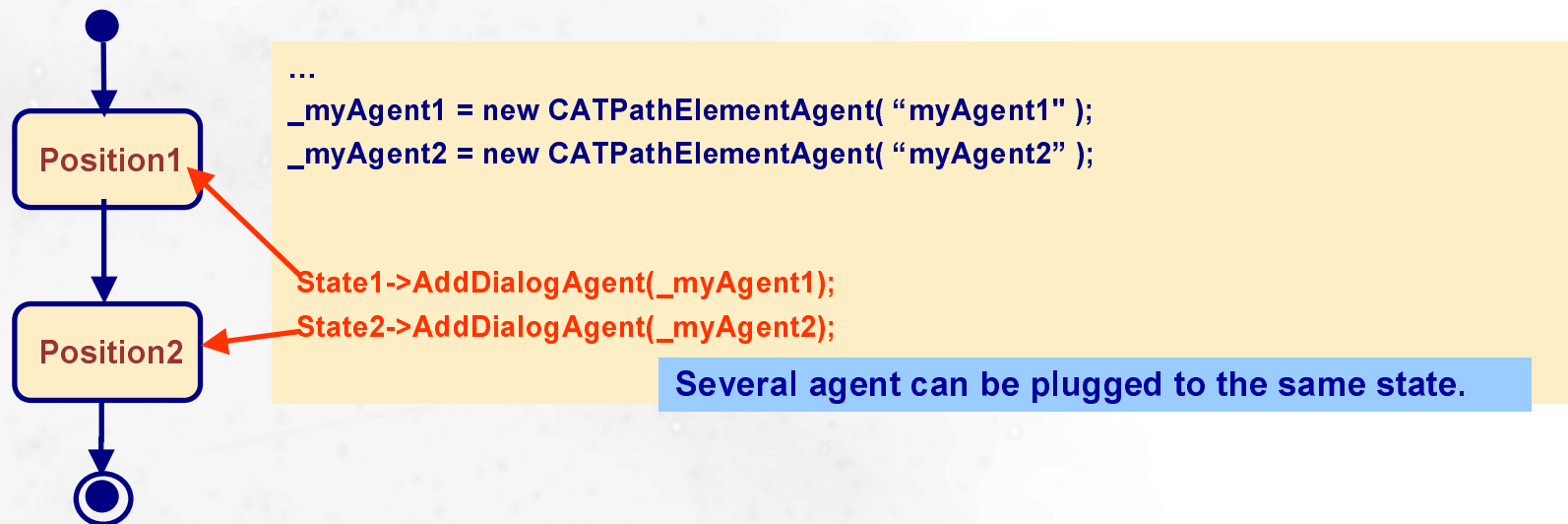
...

Enable the Object / Action dialog style.
Look in the CSO (Current Set of Objects)

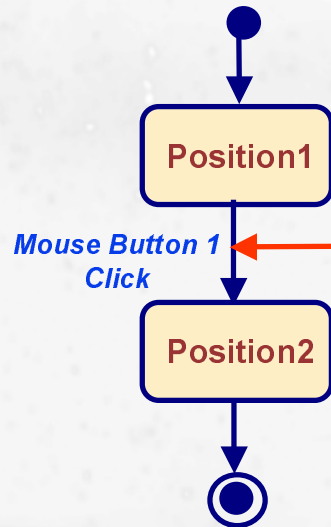
Limit the selection
to objects that implement
a given interface

A valid object is highlighted when
the mouse is located on it

Plug the Dialog Agents to the states



Define simple condition to trigger transitions



A transition is at least triggered by an end user interaction that valuates the dialog agent.

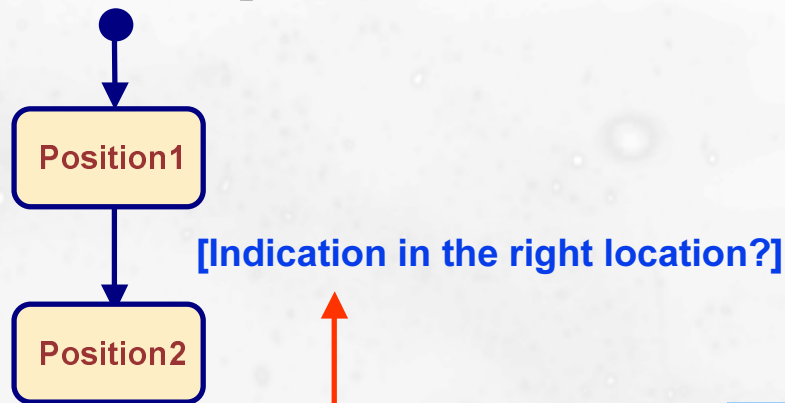
```
...
AddTransition(State1, State2,
               IsOutputSetCondition(_myAgent1), ...);
AddTransition(State2, NULL,
               IsOutputSetCondition(_myAgent2), ...);
...
```

Define condition constraining the user input



The condition method signature is:

- ***boolean ConditionMethod (void* iUsefulData)***



The transition is triggered by an end user interaction and an additional condition.

AddTransition(State1, State2,

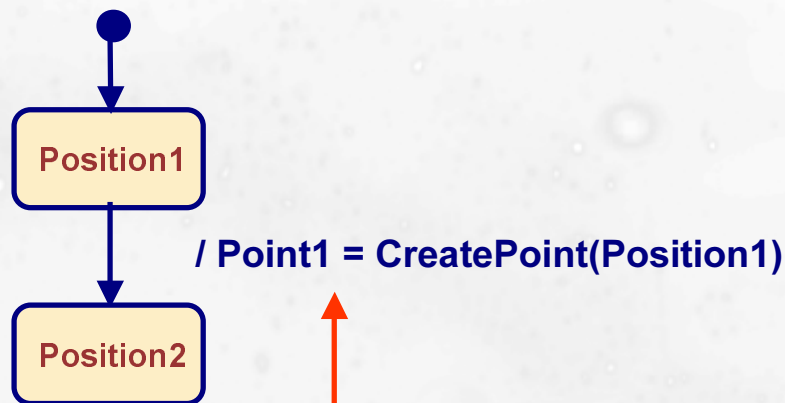
```
AndCondition(IsOutputSetCondition(_myAgent1),  
Condition((ConditionMethod)&MyCommand::CheckPosition,  
CATIPoint* PointToBeChecked));
```


Define action done in a transition



The action method signature is:

- ***boolean ActionMethod (void* iUsefulData)***



```
AddTransition(State1, State2, AndCondition(IsOutputSetCondition(_myAgent1),  
Condition((ConditionMethod)&MyCommand::CheckPosition,  
CATIPoint* PointToBeChecked)),  
Action((ActionMethod) &MyCommand::CreatePoint,  
(ActionMethod) &MyCommand::UndoCreatePoint,  
(ActionMethod) &MyCommand::RedoCreatePoint,...) );
```

Additional Information: Propose Multi Acquisition



To enable multi-selection for a path element dialog agent, set its behavior to CATDIgEngMultiAcquisition.

```
...  
_myMultiSelAgent = new CATPathElementAgent( "myMultiSelAgent" );  
_myMultiSelAgent ->AddElementType(CATISample::ClassName());  
_myMultiSelAgent ->SetBehavior(CATDIgEngMultiAcquisition);  
...
```

Enable the multi-indication or the multi-selection.
Multi-selection is possible using:

1. a trap,
2. the Search command, either run from the Edit menu or from the Power Input field,
3. the Selection Set command to reuse stored pre-selected elements

Additional Information: Retrieve Multi Acquisition



*To retrieve the selected objects, use the **GetListOfValues** method.*

```
...
CATSO* pObjSO = _myMultiSelAgent->GetListOfValues();
CATPathElement *pElemPath = NULL;
if (pObjSO)
{
    pObjSO->InitElementList();
    while (NULL != (pElemPath =(CATPathElement*)pObjSO-> NextElement()))
    {
        CAAIMyObj *pIMyObj = (CAAIMyObj *)pElemPath->FindElement(
            IID_CAAIMyObj);
        ...
    }
}
```

Enable the sort of selected elements

Additional Information: Rubber Banding



Define a self transition



Use a Dialog agent with specific behaviors

- ***CATDIgEngWithPrevaluation***
 - *The agent can be valued by the object that is under the mouse without selecting it.*
- ***CATDIgEngAcceptOnPrevaluate***
 - *The transition is triggers when the agent is valued without selecting.*



The condition to trigger the transition is:

- ***IsLastModifiedAgentCondition(_myAgent)***





After performing the corresponding action, the dialog agent needs to be recycled by:

- ***InitializeAcquisition()***

To Sum Up ...

In this Course you have seen...

-  How to define a new interactive command
-  The management of the command

Dialog

In this course you will learn the CATIA V5 infrastructure to build Dialog Boxes

- Framework objectives
- Building graphic user interfaces
- Retrieving user inputs
- The Dialog builder

Framework objectives

In this lesson you will learn the Dialog Framework objectives

Dialog Objectives

Framework objectives

Dialog Objectives

***Programmer
productivity***

Portability

***Standard
compliance***

Versatility



- ***High level objects and widgets***
- ***Promotion of reusable components***
- ***Abstract objects for:***
 - ***UNIX***
 - ***MS WINDOWS***
- ***Built on top of high level native software***
 - ***OSF/MOTIF***
 - ***MFC***
- ***Dialog applications can be run in:***
 - ***CATIA V4 - CATIA V5***
 - ***Stand alone***

Building graphic user interfaces

In this lesson you will learn what are the Dialog objects

- ▣ **Provided Objects**
- ▣ **Window**
- ▣ **Bar**
- ▣ **Menu**
- ▣ **Dialog Objects**
- ▣ **Dialog Design Steps**
- ▣ **Layout Management**
- ▣ **Resources**

Building graphic user interfaces

Provided Objects

Containers

- *Used to group, structure, and present component objects into dialog windows.*
- *Different types:*
 - ✓ *Window*
 - ✓ *Menu*
 - ✓ *Bar*
 - ✓ *Box*

Components

- *the building blocks of Graphic User Interface.*
- *May be linked to application functions*
- *Different types:*
 - ✓ *Indicator*
 - ✓ *Control*
 - ✓ *Menu Item*

Building graphic user interfaces

Window

A window is an independent resizable container in which other Dialog objects are created.

Class

Usage

CATDlgDocument

The main window for any application

CATDlgDialog

Transient window for a particular task

CATDlgNotify

Transient window for short messages

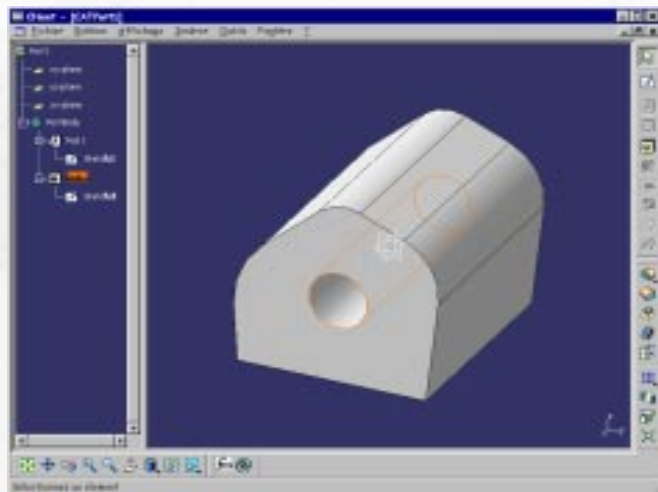
CATDlgFile

File selection window

Building graphic user interfaces

Window Types

CATDlgDocument



CATDlgDialog



CATDlgNotify



CATDlgFile



Building graphic user interfaces

Bar

A bar is a standard permanent zone for task starter or information fields.

Class

Usage

CATDlgToolBar

Zone with direct access to application tasks

CATDlgStatusBar

Zone for transient or permanent information

Building graphic user interfaces

Menu

A menu is a pop-up container for permanent task starter.

<i>Class</i>	<i>Usage</i>
<i>CATDlgBarMenu</i>	<i>Bar for main application menus</i>
<i>CATDlgSubMenu</i>	<i>Pop-up menu</i>
<i>CATDlgContextualMenu</i>	<i>Contextual menu</i>

Building graphic user interfaces

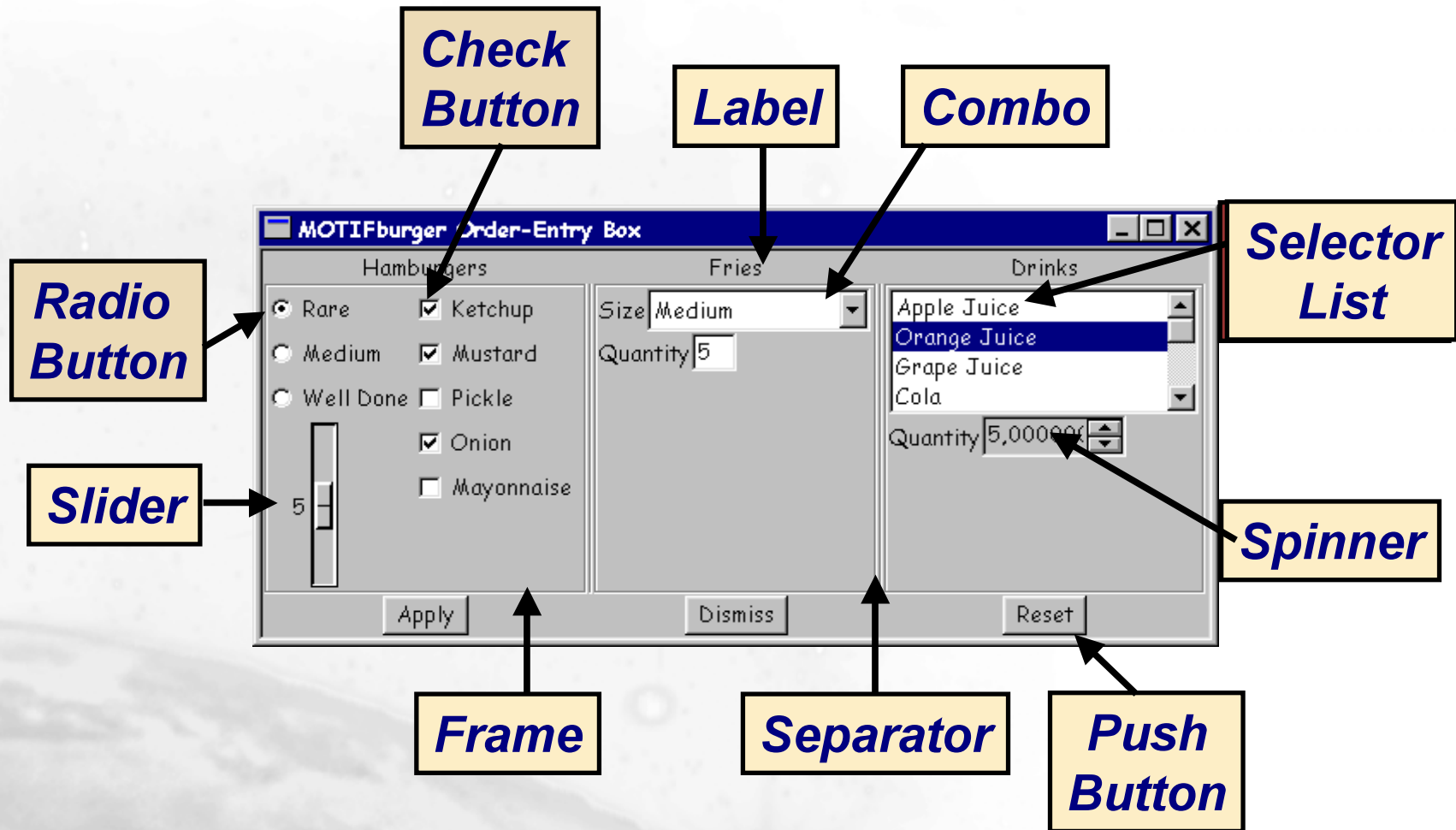
Menu Items

A menu item is a pop-up control for permanent task starter.

<i>Class</i>	<i>Usage</i>
<i>CATDlgPushItem</i>	<i>Command button in a menu</i>
<i>CATDlgRadioItem</i>	<i>Radio button in a menu</i>
<i>CATDlgCheckItem</i>	<i>On/Off button in a menu</i>
<i>CATDlgSeparatorItem</i>	<i>Separation line in a menu</i>

Building graphic user interfaces

Dialog Objects



Building graphic user interfaces

Dialog Objects : Box

A box is a container for different kinds of presentation.

Class	Usage
<i>CATDlgFrame</i>	<i>Basic container to group dialog objects</i>
<i>CATDlgContainer</i>	<i>Fixed size scrollable container</i>
<i>CATDlgTabContainer</i>	<i>Tabbed index container</i>
<i>CATDlgTabPage</i>	<i>Tabbed index page</i>
<i>CATDlgSplitter</i>	<i>Two zones separated by a sash</i>
<i>CATDlgIconBox</i>	<i>Pop-up container for icons</i>
<i>CATDlgWindows</i>	<i>MFC widget container</i>
<i>CATDlgMOTIF</i>	<i>MOTIF widget container</i>

Building graphic user interfaces

Dialog Objects : Indicator

An indicator is a passive component to enhance the presentation.

<i>Class</i>	<i>Usage</i>
<i>CATDlgLabel</i>	<i>Text or icon</i>
<i>CATDlgSeparator</i>	<i>Vertical or horizontal separation line</i>
<i>CATDlgProgress</i>	<i>Progress indicator for long operations</i>

Building graphic user interfaces

Dialog Objects : Button

A button is a single action control to start tasks or set options.

<i>Class</i>	<i>Usage</i>
<i>CATDlgPushButton</i>	<i>Command button</i>
<i>CATDlgRadioItem</i>	<i>Radio button</i>
<i>CATDlgCheckItem</i>	<i>On/Off button</i>

Building graphic user interfaces

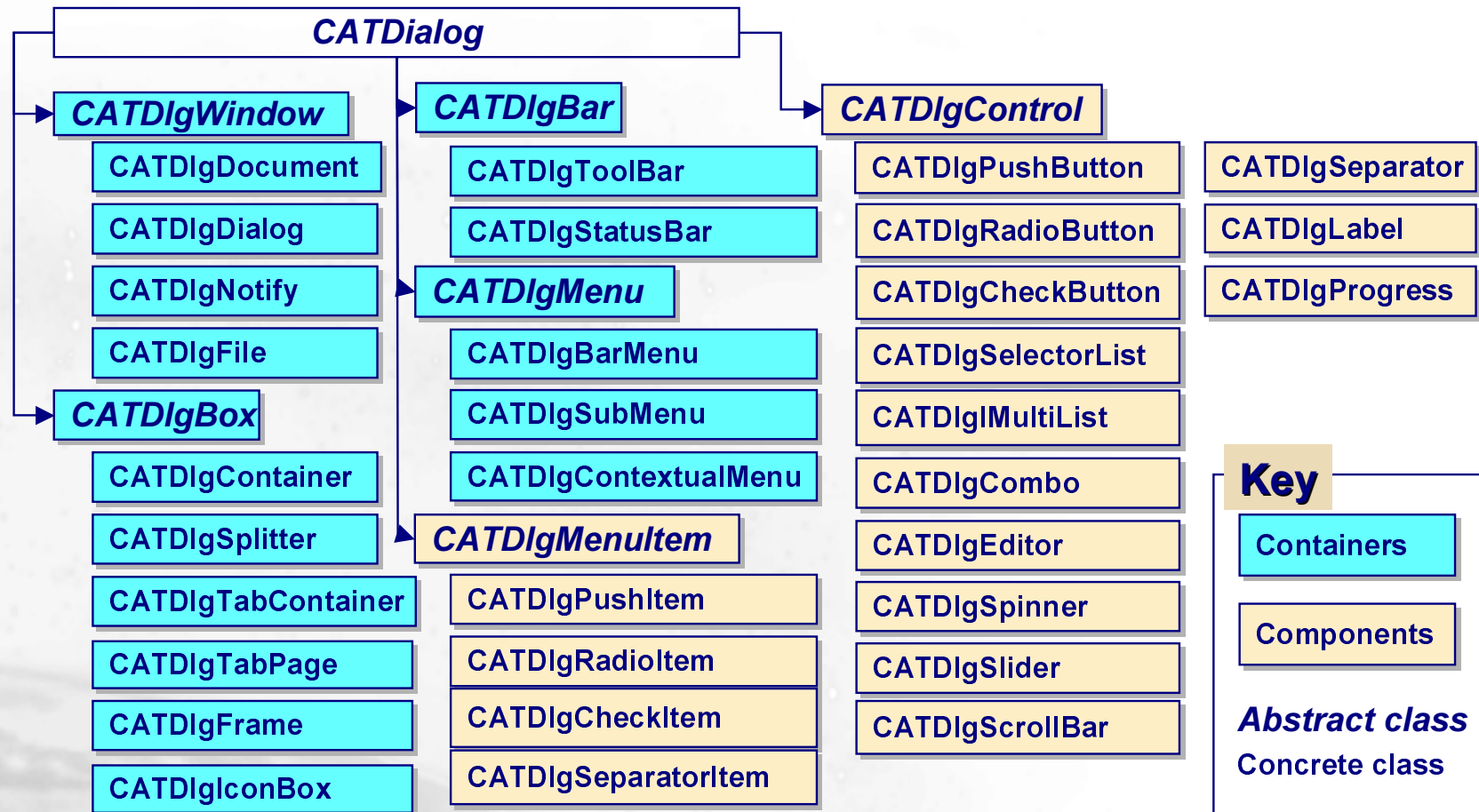
Dialog Objects : Control

A control is an object with which the end user interacts.

<i>Class</i>	<i>Usage</i>
<i>CATDlgEditor</i>	<i>Entry field</i>
<i>CATDlgCombo</i>	<i>Entry field with list</i>
<i>CATDlgSpinner</i>	<i>Numerical value selector</i>
<i>CATDlgSlider</i>	<i>Numerical value selector</i>
<i>CATDlgScrollBar</i>	<i>Scroll handler</i>
<i>CATDlgSelectorList</i>	<i>List selector</i>
<i>CATDlgMultiList</i>	<i>Multi-column list selector</i>

Building graphic user interfaces

Dialog Framework Objects



Building graphic user interfaces

Dialog Design Steps



First, determine the type of your dialog

- ***It is a dialog box or a window that contains several representations of a document.***
 - Create a class that derives from CATDlgDialog.
- ***It is a message pop-up.***
 - Instantiate the CATDlgNotify class
- ***It is a file selection box.***
 - Instantiate the CATDlgFile class.
- ***It is an application main window.***
 - Create a class that derives from CATDlgDocument.
 - Add a menu bar, a status bar, etc... (these object are provided by the Dialog framework or by the ApplicationFrame framework).

Building graphic user interfaces

Dialog Design Steps



Second, design the dialog appearance.

- ***Controls, frames, labels...***
- ***User interactions and actions.***
- ***Specifies the layout of the dialog.***



Third, implement callback methods.

- ***A callback is called in response to a user interaction with a control.***
- ***You should provide one callback per user interaction.***



Fourth, provide dialog resources.

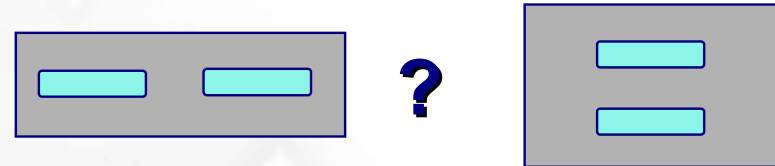
- ***Resources are text and graphics displayed in the dialog.***
- ***Using resources facilitate translation.***

Building graphic user interfaces

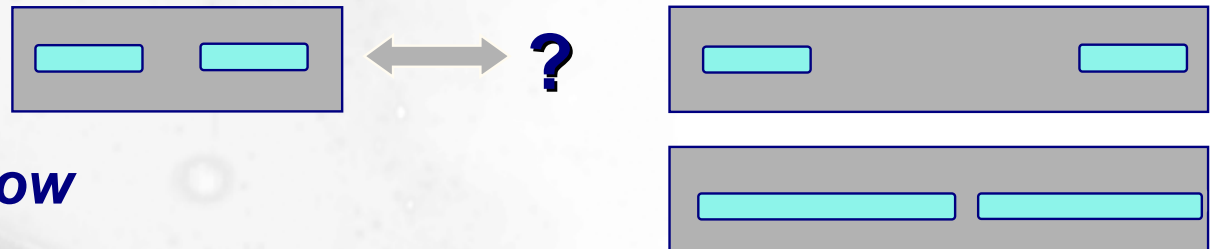
Layout Management

- *The layout defines the way the objects are organized inside a container.*

- *when the container is initially displayed*



- *when the container's window is resized*



Building graphic user interfaces

Layout Management



There are two kinds of layout management

- ***Basic layout management***
 - ***Dialog objects are dispatched in containers,***
 - ***generally sufficient to build most of non resizable windows.***
- ***Grid layout management***
 - ***Container is divided into cells, dialog objects are placed inside a cell,***
 - ***for more complex and/or resizable windows.***
 - ***Dialogs generated by the Dialog Builder use the grid layout.***

Building graphic user interfaces

Basic Layout Management



An object has a "natural" size depending on its definition parameters.



Once its children are placed, the size of a container is known.

- ***SetRectDimensions()***



Children are displayed in their container in the order in which they are created.

Building graphic user interfaces

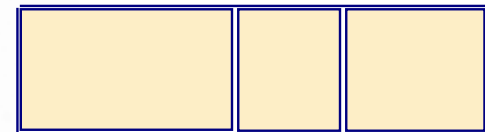
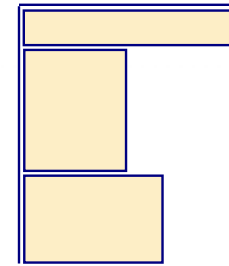
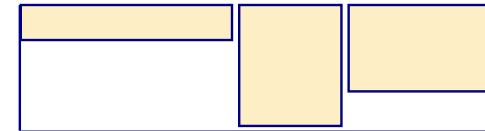
Basic Layout Management

- ***By default, all the objects of a container are arranged side by side, horizontally, justified on top, in the order in which they are created.***
- ***This default can be changed to Vertical by using the SetDefaultOrientation method:***

```
_container->SetDefaultOrientation(Vertical);
```

- ***The objects can be resized in order to fill all the room in the container by using the Attach4Sides method***

```
_container->Attach4Sides(child1);  
_container->Attach4Sides(child2);  
_container->Attach4Sides(child3);
```



Building graphic user interfaces

Grid Layout Management



For more complex cases and/or for resizable windows, the Grid Layout should be used:

- Objects are placed within a grid***

Columns	0	1
Rows	Macro	? X
	Macro Name: <input type="text"/>	Run
	E:\users\psr\Macros\Macros dans la doc\Infrastructure\C	Cancel
	E:\users\psr\Macros\Macros dans la doc\Infrastructure\C	Edit
0		Create
		Select
		Delete
1	Macro in: <input type="text" value="External File"/>	
	Description <input type="text"/>	
2	Use one of the following statements	

Building graphic user interfaces

Grid Layout Management



To use the grid layout, you need to:

- ***Create a container using the CATDlgGridLayout style***

```
__container = new CATDlgFrame(this,"MyContainer",CATDlgGridLayout);
```

- ***State where to place each dialog object inside the container***
- ***State how to attach each dialog object with respect to the cell sides***
- ***Enable rows and columns for resize.***

Building graphic user interfaces

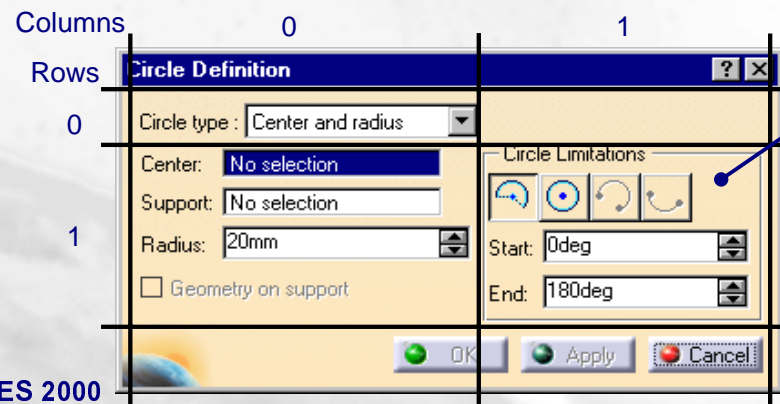
Grid Layout Management

- **To place a dialog object you can either**
 - create a *CATDlgGridConstraints* object and assign it to the dialog object,

```
CATDlgGridConstraints gridCst;  
gridCst.Row = 1;  
gridCst.column = 1;  
gridCst.H_Span = 1;  
gridCst.V_Span = 1;  
gridCst.Justification = CATGRID_4SIDES;  
pCircleLimitationsFrame->SetGridConstraints(gridCst);
```

- or set the grid constraint on the dialog object

```
pCircleLimitationsFrame->SetGridConstraints(1,1,1,1,CATGRID_4SIDES);
```



CircleLimitationsFrame:

- placed at row 1 and column 1,
- 1 row span and 1 column span,
- attached to four sides.

Building graphic user interfaces

Grid Layout Management

- ***Use the `SetGridColumnResizable` and `SetGridRowResizable` methods to specify which columns and/or rows will be sensitive to the container resize***
 - ***By default rows and column are non resizable.***
 - ***Example : first row and second column are set to be resizable.***

```
container->SetGridRowResizable(0,1);  
container->SetGridColumnResizable(1,1);
```

Column index

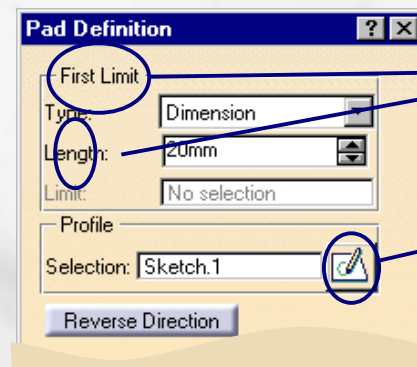
{ 0 for non resizable
1 for resizable

Building graphic user interfaces

Resources

– What are resources

- *Resources could be text or icons displayed by the application.*
- *Text resources are compatible with National Language Support*
- *Resources can be changed without recompilation*



Text resources are stored in a CATNls file

Path to Icon resource is stored in a CATRsc file

Building graphic user interfaces

Assigning Resources

– *Declare resources in your object*

MyDialogBox.h

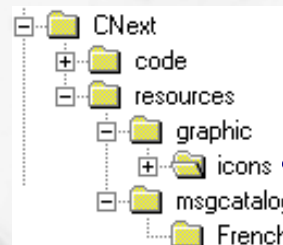
```
class MyDialogBox : public CATDlgDocument {  
    DeclareResource(MyDialogBox, CATDlgDocument);  
    ...  
};
```

Father class for
resource inheritance

Class
name

– *Create resource files*

- *One message file (CATNIs) for each supported language.*
- *One CATRsc file for non message resources.*



Icon files (*.bmp)

English files

French language support files

{ MyDialogBox.CATNIs
MyDialogBox.CATRsc

Building graphic user interfaces

Assigning Resources

– Using user defined resources

- *These are messages that you define and use by yourself.*

MyDialogBox.cpp

```
CATUnicodeString text = CATMsgCatalog::BuildMessage("MyDialogBox", "Pick");
```

Message Key

Message File

MyDialogBox.CATNls

```
Pick = "Indicate coordinates";
```

– Using predefined resources

- *Every dialog object have some predefined resources, like title, help and icon.*
- *To use them, declare them in your CATNls or CATRsc file.*

MyDialogBox.CATNls

```
Title = "A simple panel";
```

Building graphic user interfaces

CATDialog Predefined Resources

- **Text Resources**

- *Title : object title string*
- *Mnemonic : Alt key shortcut to select a displayed menu or menu item*
- *Accelerator : Ctrl key shortcut to select a menu item*
- *Help : help message associated with the object*
- *ShortHelp : short message displayed when the mouse stays over this object.*
- *LongHelp : message displayed when the user clicks on Help button or on the ? box, the cursor becoming a ?, and clicks on the object to get help about it.*

- **Icon Resources**

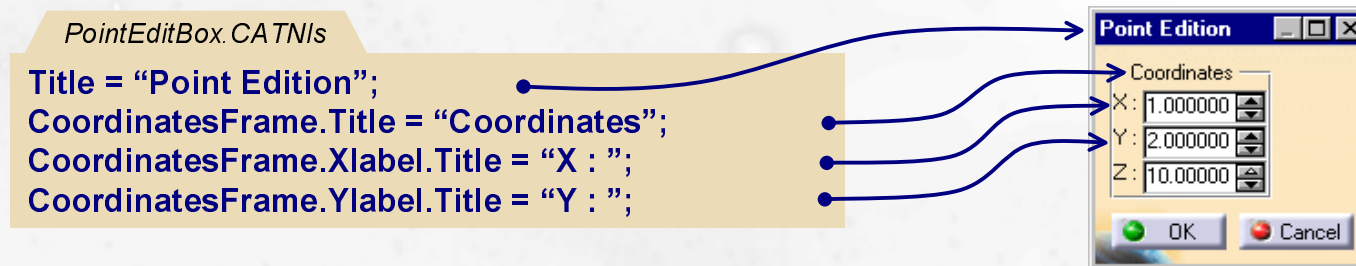
- *Icon : dialog object default icon*
- *IconSel : icon displayed when this object is selected.*
- *IconFocus : icon displayed when the mouse moves over this object.*
- *IconDisabled : icon displayed when this object can not be selected.*
- *IconType : type of the icon (Default, General, Creation, Modification, or Analysis), used to set a background color if the icon is transparent.*

Building graphic user interfaces

Concatenation and Inheritance

– Concatenation

- Resources of contained objects can be defined in the container object resource file.



– Inheritance

- Derived object inherit from any resources defined in the father object.

Retrieving user inputs

In this lesson you will learn how to associate Commands to your Dialog

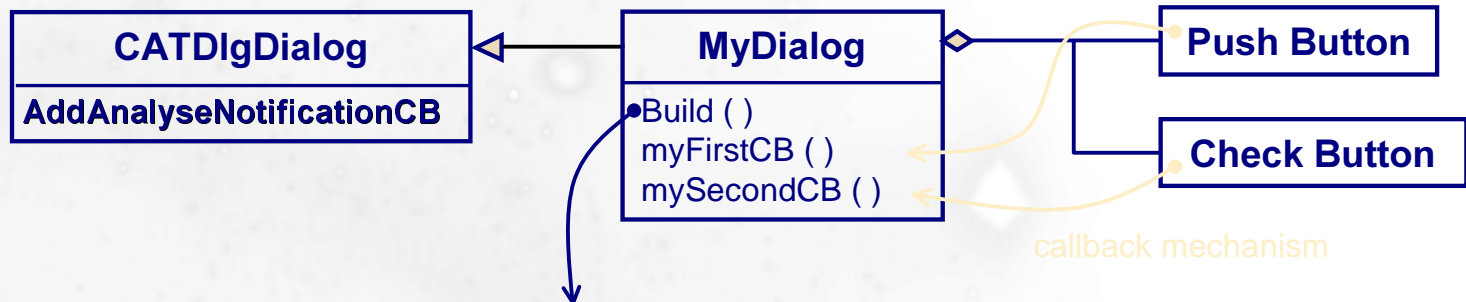
- ▣ Defining a callback
- ▣ Dialog Engine Integration
- ▣ Application Architecture

Retrieving user inputs

Defining a callback



Callbacks are set with the *AddAnalyseNotificationCB* method



Control object that will callback
the current object

Notification

CB method

User data to be passed
to the CB method

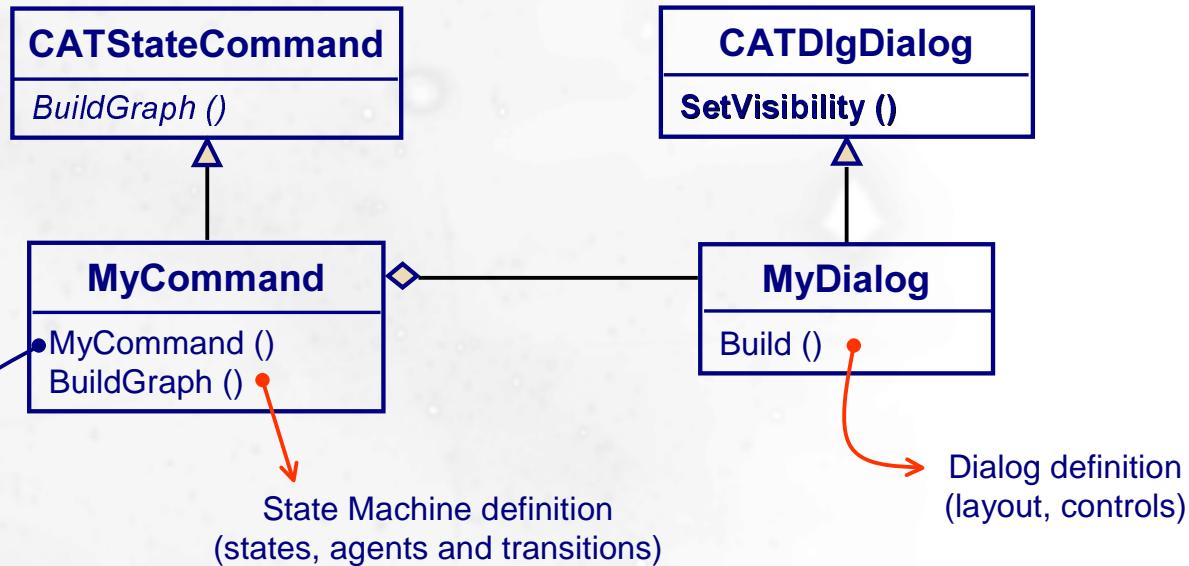
```
CATDlgPushButton *pOKButton;  
pMyButton = new CATDlgPushButton(this, "Apply");  
AddAnalyseNotificationCB (  
    • pMyButton,  
    • pMyButton->GetPushBActivateNotification(),  
    • (CATCommandMethod)&MyDialog::myFirstCB,  
    • NULL);
```

Retrieving user inputs

Dialog Engine Integration



Associate a dialog to your command



```
panel = new MyDialog();
panel->Build();
panel->SetVisibility(CATDlgShow);
```

Provided by Dialog framework
To be overloaded by application programmer
Provided by application programmer

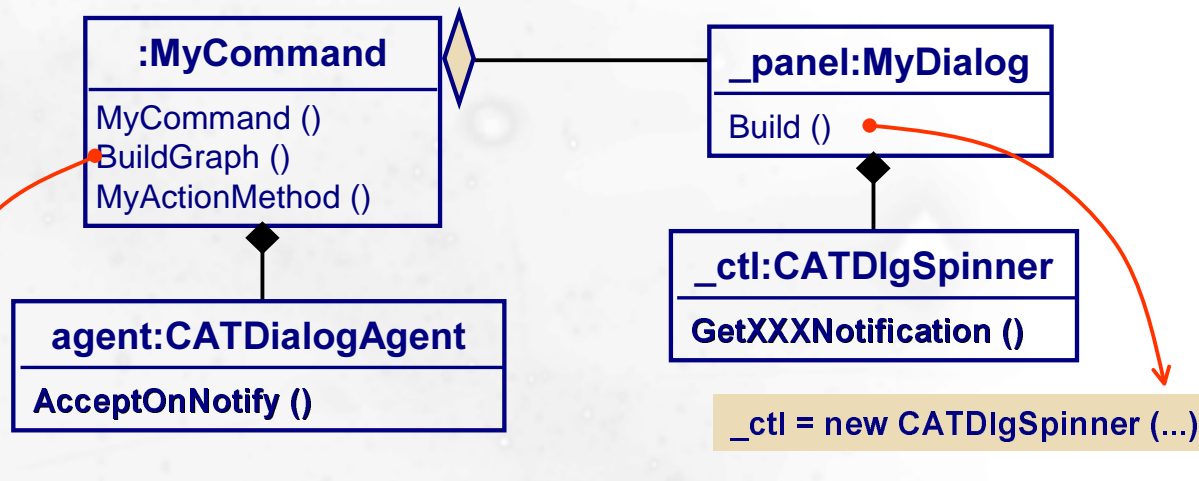
Key

Retrieving user inputs

Dialog Engine Integration



Use a DialogAgent instead of a callback when you want to perform a transition.

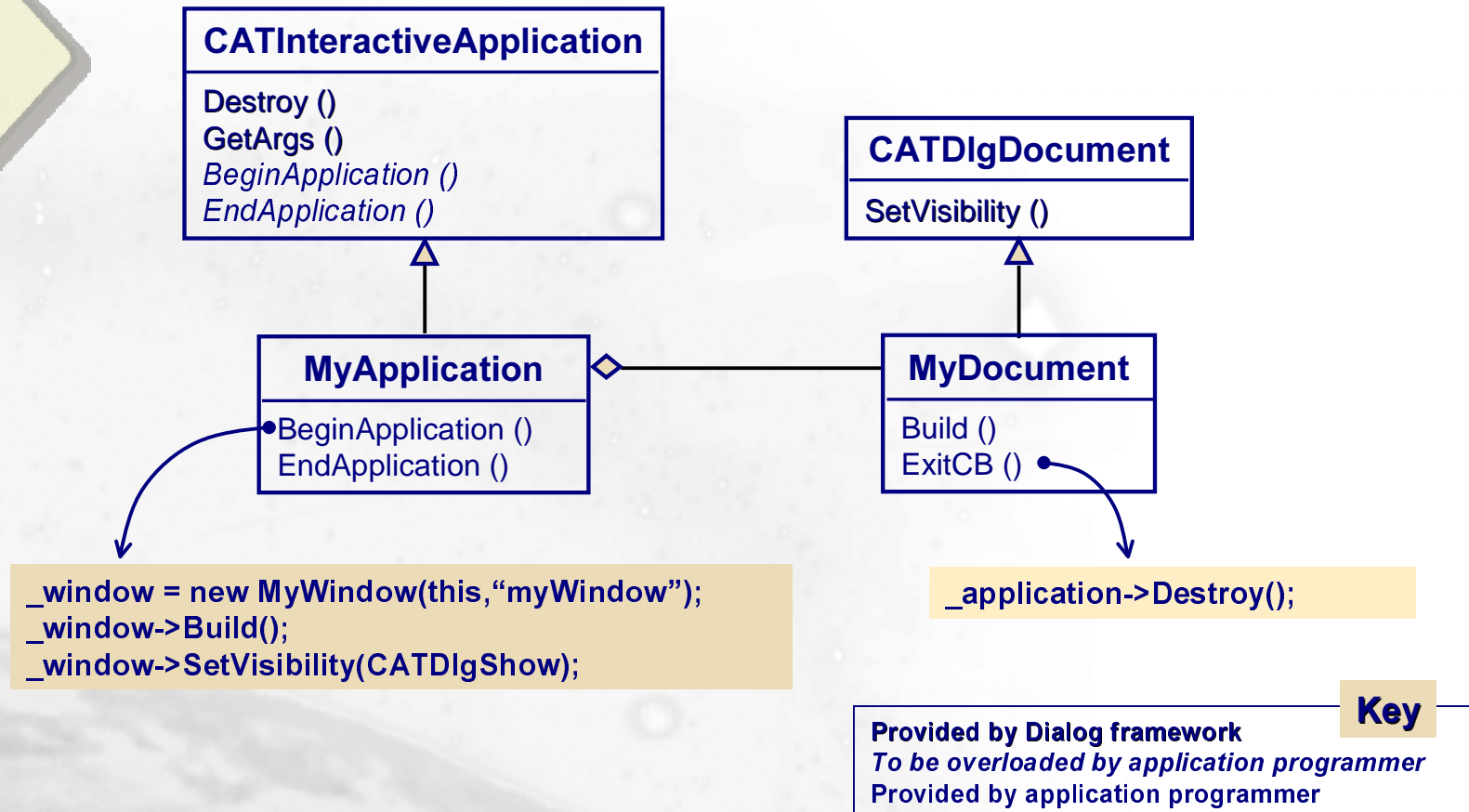


```
agent = new CATDialogAgent("Agent");
agent->AcceptOnNotify( _panel->_ctl, _panel->_ctl->GetXXXNotification());
firstState->AddDialogAgent(agent);
...
AddTransition (firstState, secondState, IsOutputSetCondition(agent),
Action((ActionMethod)&MyCommand::MyActionMethod));
```

```
_ctl = new CATDlgSpinner (...);
```


Retrieving user inputs

Application Architecture



The Dialog builder

In this lesson you will learn the CATIA V5 wizard to define your Dialog

Dialog Builder

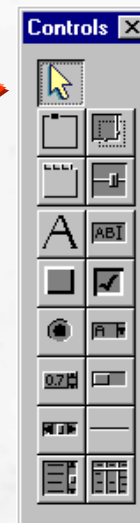
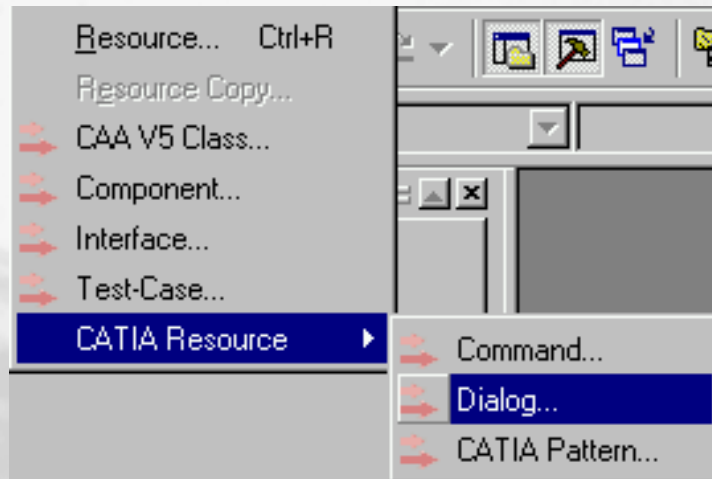
The Dialog Builder

Dialog Builder

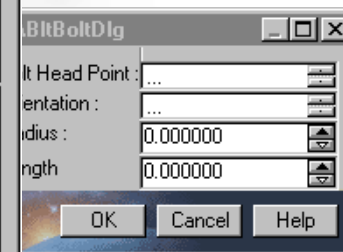


CATIA Dialog Builder facilitates panel creation and edition.

- Access through menu or by double-clicking on an existing *.CATDlg file.
- Generates C++ code.
- Possibility to define Callbacks.



Dialog Builder Toolbar



To Sum Up ...

In this Course you have seen...

- **All the Dialog Objects provided by CATIA V5**
- **How to associate Commands to the Dialog**
- **The Dialog Builder**

CAA V5 Administration

You will learn what are the prerequisites to install CAA on a workstation and how to manage the applications build with CAA

 **Packaging**

 **Licensing**

 **Software Prerequisites**

 **Delivering a CAA build Application**

CAA V5 Physical Packaging



Rapid Application Development Environment (one CD)

A set of programming tools

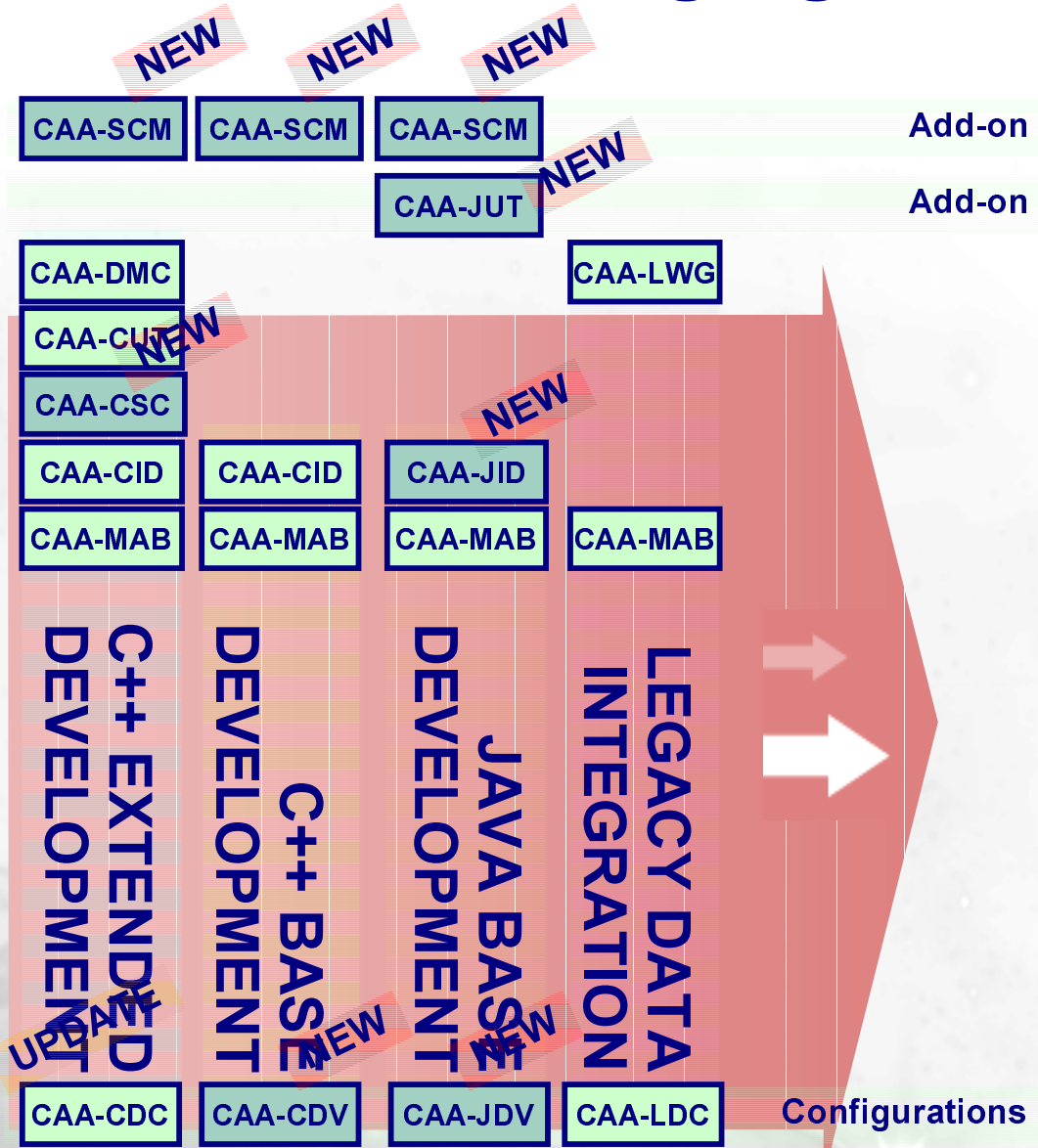


V5 API for CATIA (one CD)

A set of headers with their corresponding documentation

The CAA V5 Encyclopedia

CAA V5R7 Packaging and Portfolio



- ✓ Customization with
 - CATIA
 - ENOVIA server
 - ENOVIA 3d com
- ✓ Data Federation with:
 - ENOVIA 3d com
 - WEB browser

CAA V5 RADE (for CATIA) includes



C++ Interactive Dashboard Product (CID)

Provide developers with an environment for building C++ Component Application Architecture (CAA) applications

Tight integration with Microsoft Visual Studio C++



C++ Interactive Dashboard Product is only available on Windows NT and gives access to the other RADE products.

Multi-Workspace Application Builder Product (MAB)

Provide a single tool to compile and link applications whatever the programming languages used (Fortran, C, C++, Java)

Available both on NT and UNIX



C++ Unit Test Manager Product (CUT)

Enables users to check development compliance with design scenarios and to ensure regression-free modifications, scenarios pertinence, automatic result comparisons, timeout performance replay

Only on Windows NT : Integration with Rational Purify for memory management tests and Rational Pure Coverage for test coverage

CAA V5 RADE (for CATIA) includes



Data Model Customizer (DMC)

UML modeling authoring for ENOVIA object model & CATIA object model

ENOVIA Modeling object optimized publication capability

Offer same user interfaces for CATIA and ENOVIA object extension



C++ Source Checker (CSC)

Automatic check of C++ V5 coding rules

Memory leaks debugging

C++ source parser and full HTML report



Source Code Manager (SCM)

Workspace management

Collaborative and integrated code distribution

Concurrent development support

Version and configuration control

Multi-platform workspace management

Scales from small teamwork to large development organization

CAA V5 Licensing Mechanisms



CDC or CDV license is to use CAA V5 RADE (C++)

CAA V5 can be used in two licensing modes, either nodelock or with concurrent usage of licenses on a network, just like standard DS products.



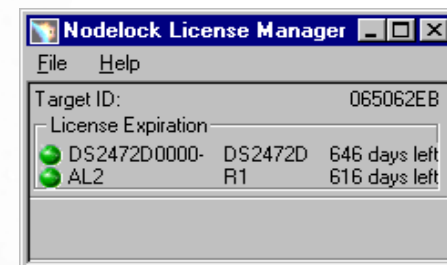
Licensing mechanisms are based on LUM (License Use Management)

Windows NT workstations must have a LAN Card (Ethernet or Token Ring) and TCP/IP installed and properly configured, even in the case of nodelock licensing



On NT : Use Nodelock Key Management

Select the Start->Programs->CATIA->Tools->Nodelock Key Management command to check Nodelock licenses if you use Nodelock licenses.



CAA V5 Software Prerequisites (1)



C++ Interactive Dashboard Product (CID) requires

Microsoft Visual C++ 6.0 product SP3

*The Unicode option **MUST** be selected when installing Visual C++*

Microsoft Internet Explorer (delivered with Windows NT 4.0), at minimum level 4.01 Service Pack 1



Multi-Workspace Application Builder Product (MAB) requires:

Some specific compiler level depending on the OS (cf. following foils)

JAVA JDK 1.1.6 or JAVA JDK 1.1.8 when using IDL compiler

CAA V5 Software Prerequisites (2)



C++ Unit Test Manager Product (CUT) requires

For Automatic Run time batch test replay : MKS Toolkit® V6.1 on Windows NT

For Automatic memory management check : Rational Purify V6.5

For Automatic test coverage computation : Rational Pure Coverage V6.5



Access to Online Documentation requires an HTML browser

In a UNIX environment : Netscape Navigator at minimum level 4.5

In a Windows environment, either Microsoft Internet Explorer at minimum level 4.01 Service Pack 1 or Netscape Navigator at minimum level 4.5

CAA V5 Software Prerequisites (3)



Windows NT Environment

Microsoft Windows NT Workstation Version 4.0 with Service Pack 4 or 5 or 6a

Microsoft Visual C++ 6.0 product SP3



IBM AIX Environment

AIX Version 4 Release 3.3

IBM C and C++ for AIX Compilers Version 3.6.4 or 3.6.6 (ibmcxx).

CAA V5 Software Prerequisites (4)



SGI IRIX Environment

IRIX 6.5.2m

C, C++, MIPSpro Compiler 7.2.1 (n32 ABI)



Sun Solaris Environment

Sun Solaris 2.6.0 or Solaris 7

C, C++, SUN WorkShop Compilers 4.2



HP-UX Environment

HP-UX Version 10.20 A.C.E. 4

C compiler A.10.32.03

C++ compiler aC++ A.01.21

CAA V5 Installation Procedure



Check the CATIA installation

You need to have the same Release and Service Pack as for CAA

*The directory path for CATIA **cannot** contains blank characters*



Install CAA V5 API for CATIA

The V5 API are installed in the same directory than CATIA

Install first the GA version then the proper Service Pack



Install CAA V5 RADE

Choose a separate directory to install the CAA V5 RADE product

Install first the GA version then the proper Service Pack

Delivering an Application (1)



Copy the Runtime View of your CAA application.

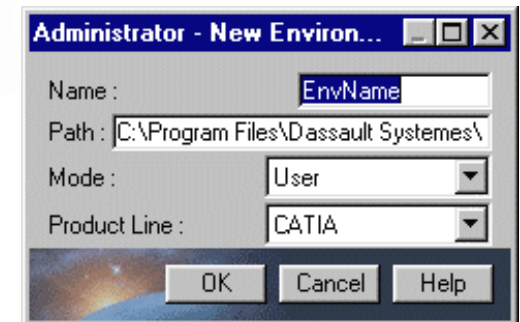


On NT only : use Environment Editor to create a new environment

Select the Start->Programs ->CATIA ->Tools->Environment Editor command

In the Dialog Box provide :

- ***A name for the new environment***
- ***The path***
- ***The mode (user / global). The environment will be available for all the users or only for the current one***
- ***The product line (CATIA)***



***The path has to reference CATIA and the Runtime View of your CAA application
c:\catia_v5\intel_a;c:\myApplication\intel_a***

Delivering an Application (2)



***On NT and on UNIX :
use the “setcatenv” command to create a new environment***

*The setcatenv command is located in \Install_folder\OS_a\code\bin
Install_folder is the directory where CATIA is installed
OS_a can be intel_a, aix_a, hpux_a, irix_a, solaris_a*

The main options are :

- e : to specify the name of the new environment*
- d: to specify the directory where the CATIA environments are stored*
- p : to give the path*

*The path has to reference CATIA and the Runtime View of your CAA application
setcatenv -e myEnv -d c:\catia_v5\CATEnv -p c:\catia_v5;c:\myApplication*

*If the path contains a blank character, use double quotes (“”)
setcatenv -e myEnv -p "c:\Program Files\Dassault Systemes\B06;c:\myApplication"*

Delivering an Application (3)



When you create a new environment :

The set of variables is defined

A new shortcut is created on the desktop to start CATIA with the new CAA Application

*A new command is created in the Start menu (NT only)
Start-> Programs-> CATIA -> myEnv*



You can deliver more than one CAA Application

In the path, you can provide several directories

setcatenv -e myEnv -p c:\catia_v5;c:\myApplication1;c:\myApplication2



To delete an environment use delcatenv

Usage : delcatenv -e myEnv

To Sum Up ...

In this lesson you have seen...

- The prerequisites to install CAA V5 on a workstation
- How to deliver a CAA V5 application to a customer