SIDEXIS neXt Generation

GBE

Programmer's Guide

SIDEXIS XG Programming Model Version 1.2

Programmer's Guide Version 1.2 (Released)

History

Date	Revision	Name
26.09.01	Draft	M. Reinke, RSWE
09.11.01	Review modifications	M. Reinke, RSWE
18.03.03	SIDEXIS XG Update => Version 1.1	M. Reinke, RSWE
15.07.03	Release update => Version 1.2	M. Reinke, RSWE
15.03.04	Review & Release Version 1.2	JüZi

Product History

Date	Product Version	Name
21.03.03	SIDEXIS XG Version 1.12	JüZi
08.04.04	SIDEXIS XG Version 1.2	JüZi

1 Contents

2 Figures

3 Getting Started

3.1 About this document

This starting manual contains basic articles about general aspects in programming SIDEXIS. Refer to this documentation to get a high-level SIDEXIS programming overview.

To get more in detail please refer to the assoicated reference manuals.

3.2 References

[1] SIDEXIS XG Programmers Guide (this document)

[2] SIDEXIS XG Programmers Reference

[3] SIDEXIS XG SDK license agreement (License.txt)

3.3 Glossary

SIDEXIS XG	SIDEXIS neXt Generation
PM	Programming model
PlugIn	Dll or EXE extension module for SIDEXIS
COM	Component object model
DirectDental	SIDEXIS XG's COM interface standard
SDK	Software development kit
MSVC	Microsoft Visual C++
VBA	Visual Basic for Applications

3.4 What is SIDEXIS neXt Generation?

SIDEXIS neXt Generation ist based on a series of components and services. This architecture can be extended by adding specialized components, called PlugIns.

SIDEXIS is a modular system of pluggable components. SIDEXIS PlugIns are embedded in the SIDEXIS programming model which both offers internal SIDEXIS services to the plugins and requires some fomal plugin interface standards. The kind of services offered to the plugins are described in the SIDEXIS programming model (PM). The requested interfaces follows the DirectDental standard.

SIDEXIS is based on the Microsoft component object model (COM). As a SIDEXIS PlugIn writer some knowledge about programming COM is required.

This guide supports the plugin development process by offering an overview picture covering the top level aspects.

For obtaining details, the following reference documents should be consulted:

Document name	Contents
SIDEXIS XG Programmer's Guide [1]	This document.

Document name	Contents
SIDEXIS XG Programmer's Reference [2]	Reference covering the XG programming model in detail. The programming model is available to PlugIns and VBA scripts as well. Reference covering all aspects of DirectDental PlugIn development.

3.5 What is a PlugIn?

The range of functionality of an existing SIDEXIS installation can be extended according to the specified application by PlugIns. After one-time installation of such a module the new functionality is available to the user integrated into the SIDEXIS user interface, for example as a new button in a SIDEXIS toolbar.

PlugIns run fully integrated inside the SIDEXIS application; there is no switching into other applications necessary. Here SIDEXIS is working as a universal platform for volatile extensions that are brought into action by the user in a well-known environment for raising productivity and extending the solution spectrum. On account of the well-known SIDEXIS environment the effort for training for users is remarkably low: Just the functionality added by the new PlugIn has to be learned, the SIDEXIS frame is already known!

PlugIns are offered as Windows DLL (Dynamic Link Libraries) or EXE modules (Executables) and are given access to the SIDEXIS-internal Programming Model via the DirectDental interface. With this technology the user commands about flexible, safe and performing SIDEXIS extension possibilities in order to match working with SIDEXIS even better with local requirements.

A toolkit for developing own PlugIns enables you too to extend SIDEXIS. With the SIDEXIS XG SDK you obtain all necessary information, tools and samples for the creation of SIDEXIS-compatible PlugIns.

3.6 What Background Do You Need?

The amount you need to know for a particular SIDEXIS task depends on the task. For example, you must understand basic COM principles and C or C++ to create a SIDEXIS PlugIn.

The following tables show what you might need to know to perform different tasks and get the most out of the documentation.

Section	Background
Getting Started	None
A SIDEXIS PlugIn in action	Basic knowledge about SIDEXIS XG's modes of operation
Developping SIDEXIS PlugIns	Basic knowledge about COM programming.

3.7 Documentation Roadmap

To find out what background different tasks require, see What Background Do You Need?

To help you find the information you need, the following list describes the content of each section in the SIDEXIS documentation and when you will typically use it.

Section	Content
Getting Started	This chapter gives general information about SIDEXIS. Read this

Section	Content
	section to orient yourself when first starting with SIDEXIS and it's programming capabilities.
The SIDEXIS XG Architecture	Describes the overall system architecture and it's building blocks. Refer to this chapter to get an idea where your programming activities take place.
Developing SIDEXIS PlugIns	Contains a brief description of the SIDEXIS PlugIn programming process. Refer to this section to get information how to start your custom PlugIn development.

3.8 Recommended reading

To get more detailed information about several aspects of COM PlugIn programming the following reading list might be helpful:

Reference	Description
MSDN	Microsoft Developer Network, either CD-ROM distribution or Website
MSDN article Q131086	SAMPLE: SAFEARAY: Use of Safe Arrays in Automation
Inside Distributed COM, G.Eddon, H.Eddon, MS Press 1998	Covers COM and DCOM aspects

3.9 A SIDEXIS PlugIn in action

This section shows the integration and usage of some sample filter PlugIns inside SIDEXIS.

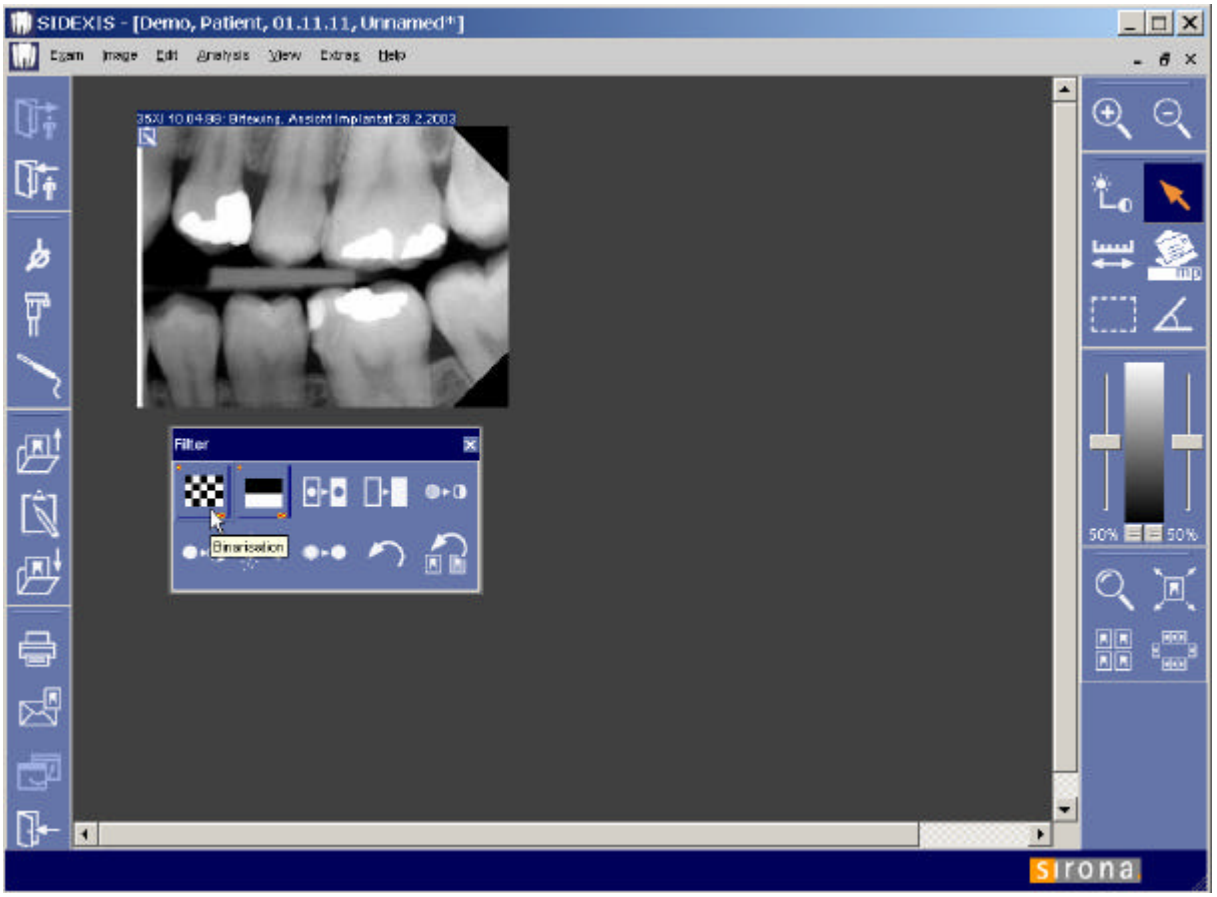

Figure 1 PlugIn selection using a custom toolbar

Figure 1 shows a typical image processing scenario using a filter provided as PlugIn. The example filter is a simple BINARIZE image processing filter.

Figure 2 Binarize filter parameter dialog

Right after the binarize filter selection a custom filter parameter dialog is shown (Figure 2). The binarize filter asks for one single grey level parameter, which represents the filter threshold: below this value all pixels are painted black, above all pixels are shown white.

After prompting the filterparameter dialog the filter operation is launched and the result will be displayed in SIDEXIS as shown in Figure 3.

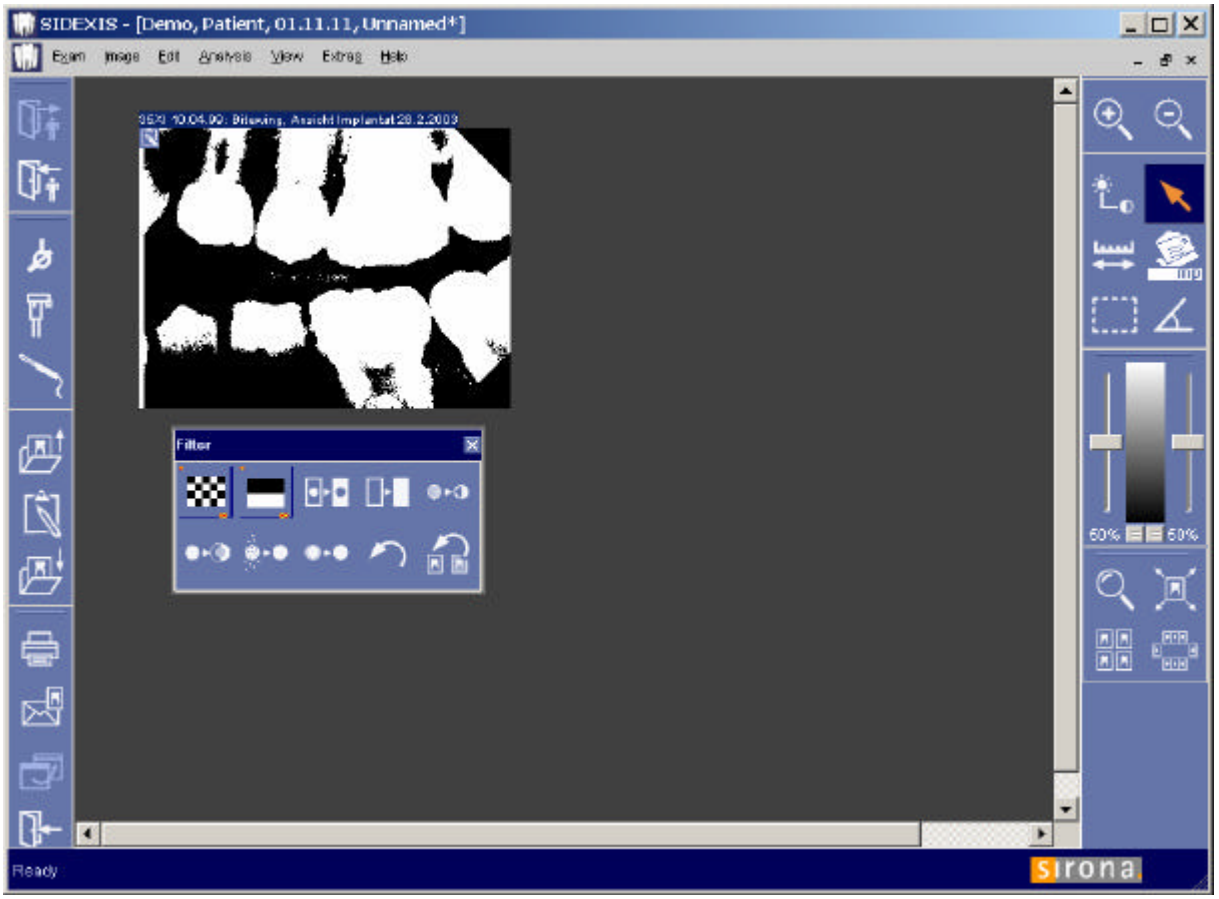

Figure 3 resulting filtered sample image

The storage of filtered images is integrated seamless into SIDEXIS, so all undo and view regeneration operations are supported.

Pressing the undo button will regenerate the unfiltered image.

So how did we get the BINARIZE PlugIn running?

First of all this filter is provided as DirectDental compatible PlugIn. The associated Dll is installed using the PlugIn Manager which is part of SIDEXIS XG's central customize dialog.

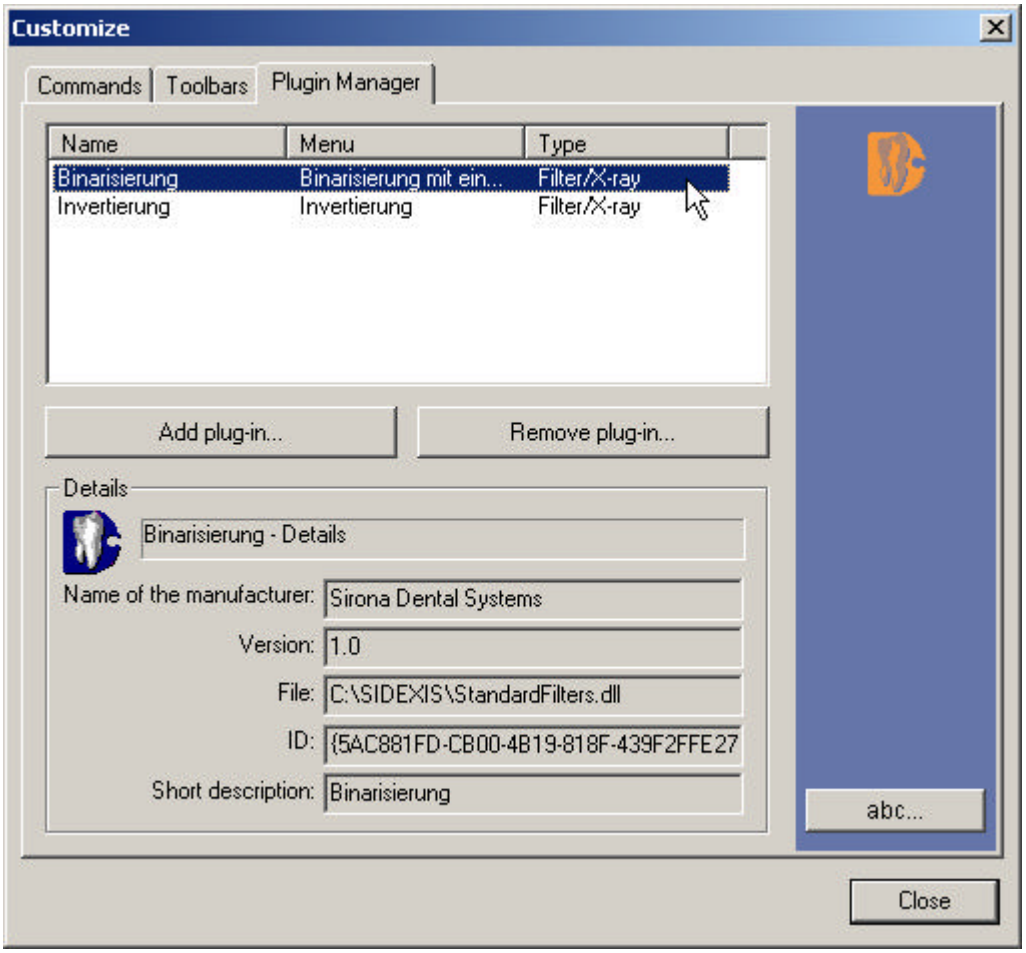

Figure 4 Filter installation via PlugIn Manager

After installation via "Add plugin..." the new PlugIns are documented in the list of available PlugIns.

Now it's time to move on to the "Commands" section of the Customize dialog. In case of previously installed filter PlugIns the new functions are presented in the "Filter" category. Selecting this option shows up all available filter PlugIns. In our case the BINARIZE Filter may now be selected via mouse and moved to any open SIDEXIS toolbar via drag&drop.

That's it. Quite easy, isn't it?

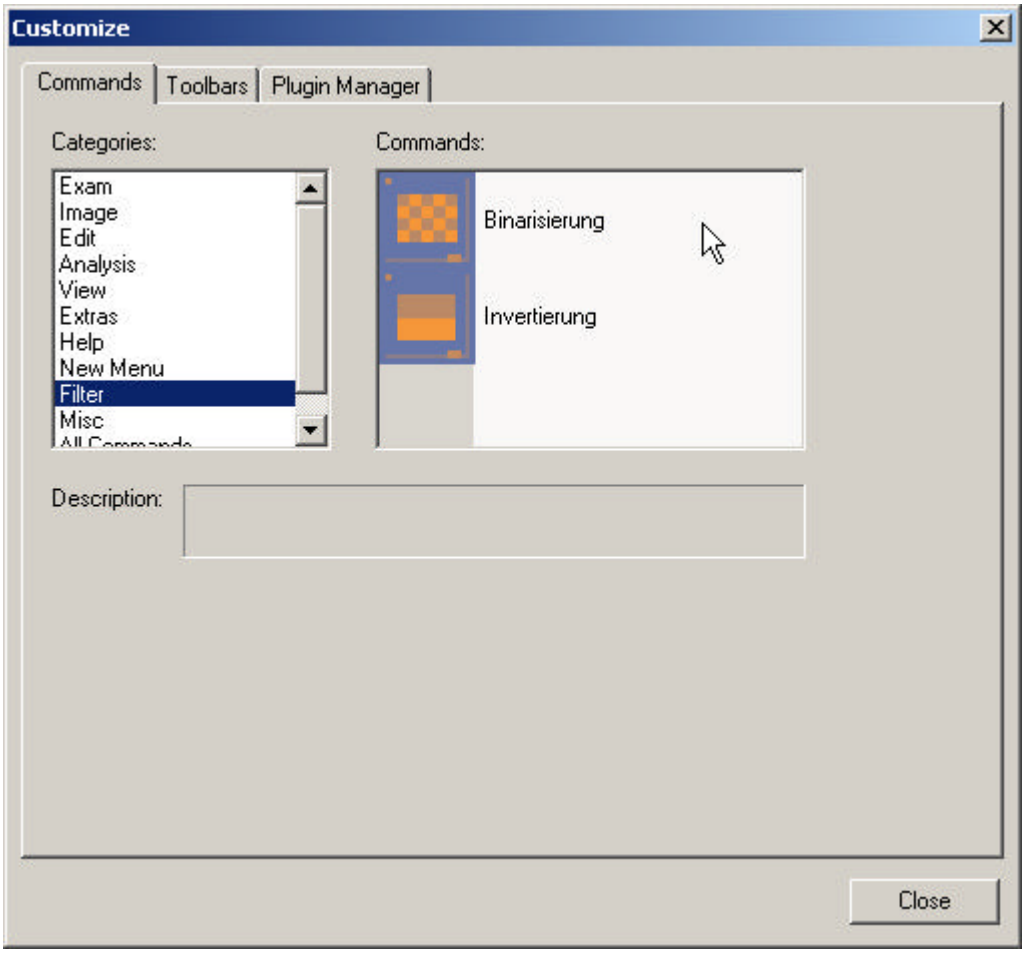

Figure 5 Filter placement via Customize dialog

3.10 Potential applications

The provided samples cover only the top of the iceberg of possible SIDEXIS extensions. Using the SIDEXIS concept any image manipulating filter can be integrated into SIDEXIS today.

In the future the SIDEXIS concept will be extended not only to filtering extensions but in the areas of file I/O, image aquisition etc.

The http://www.sidexis.com website will provide listings about available and new PlugIns both from Sirona and from independent 3rd party developers.

4 The SIDEXIS XG Architecture

This chapter offers a brief overview about the new open and extendable application architecture.

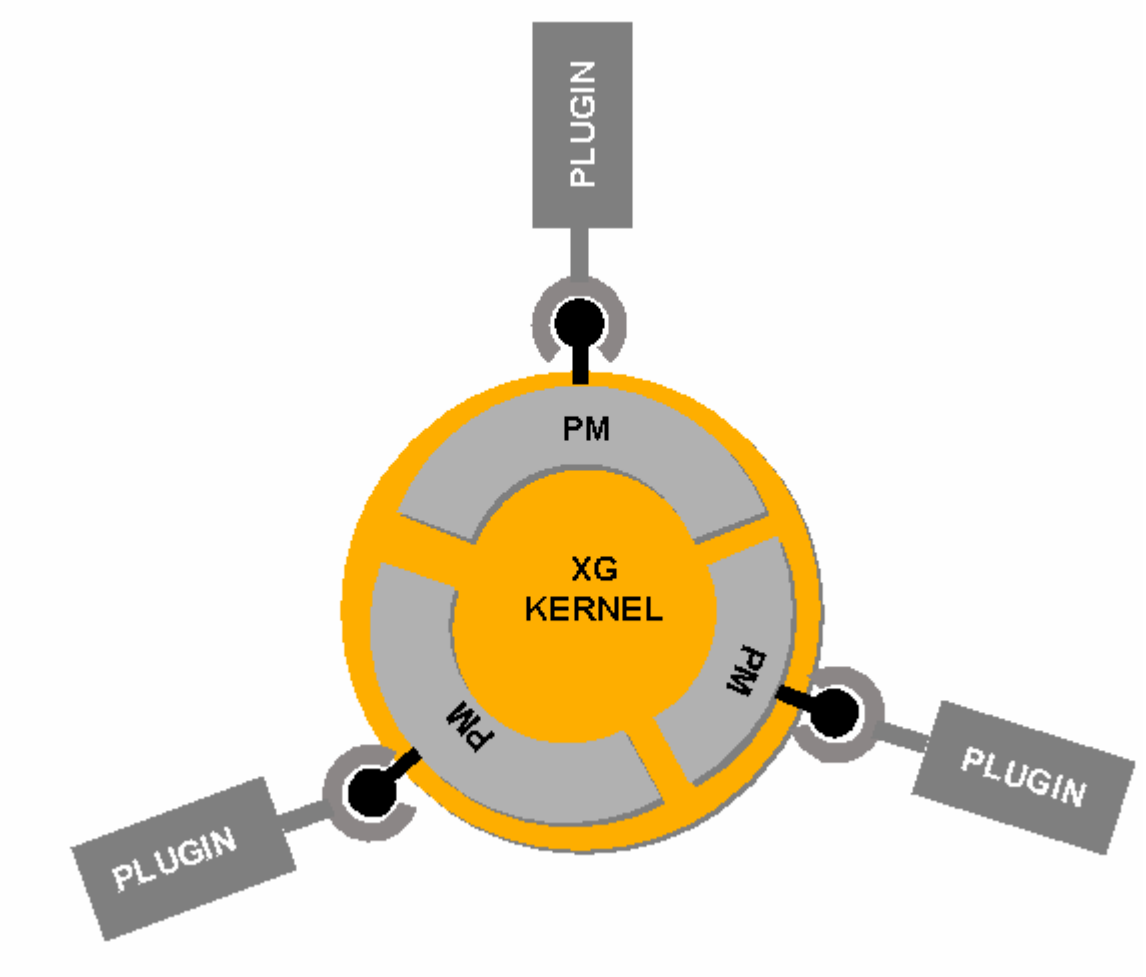

Figure 6 Plugin - PM relationship

PlugIns connect to a running SIDEXIS instance using the DirectDental interface standard. Doing so, they get hosted by SIDEXIS and are instantiated through user interaction. SIDEXIS communicates to the PlugIn using a predefined *IDirectDental* API which all PIs have to implement.

The PlugIns themselves can communicate with SIDEXIS through the XG programming model (PM). The PM covers all aspects of internal examination, image etc representation. The PM is reachable through a well documented set of objects, methods and properties.

Please refer to [2] for detailed information concerning the PM.

5 Developing SIDEXIS PlugIns

5.1 The PlugIn Interface

A SIDEXIS PlugIn is a COM EXE module which can be added to an existing SIDEXIS installation. After proper component registration through SIDEXIS's PlugIn Manager the custom new functionality is presented to the SIDEXIS user.

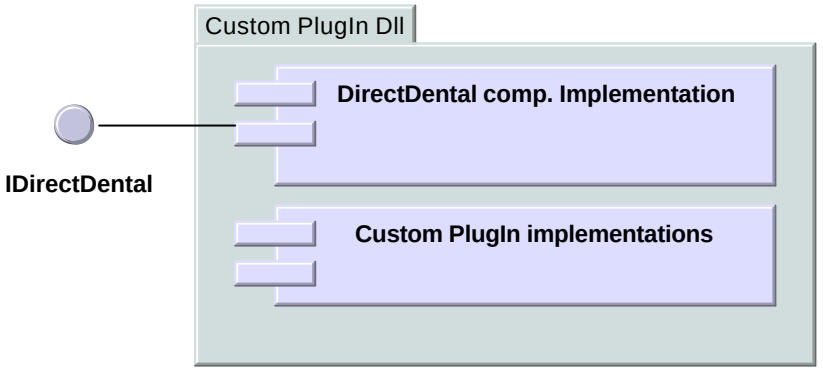

Figure 7 PlugIn deployment

5.2 Which PlugIn category should I choose?

Is your PlugIn an image processing filter…

…that can handle any image?

- Then choose category "Filter".

…that can handle greyscale images only?

- Then choose category "Filter/X-ray".

…that can handle true colour images only?

- Then choose category "Filter/Video".

Does your PlugIn something else?

- Then choose category "Void".

5.3 How do I create a SIDEXIS PlugIn?

SIDEXIS-compatible PlugIns can be developped using any environment capable to generate inprocess COM servers (= COM .DLLs) or out of process servers (=COM .EXEs). The shipped samples and sources are based on Microsoft Visual C++. This is the reference environment, but it shouldn't be a problem to get your custom PlugIns done with Delphi, VB etc.

There are two development strategies:

1. Developping from scratch using the SIDEXIS COM specifications (Refer to [2] for a detailed description of SIDEXIS COM interfaces specifications),

and/or

2. Using the Visual C++ SIDEXIS XG PlugIn Wizard provided with this SDK.

5.3.1 Custom development of SIDEXIS PlugIns

Take this advanced direction to develop custom PlugIns if you are using a non MSVC/C++ development environment (Visual Basic, Delphi etc.) or you prefer native COM interfaces of the SIDEXIS programming model.

If you are not using MSVC/C++ for your PlugIn development purposes please refer to your software development toolkit COM references for detailed information. These toolkits provide proprietary mechanisms for handling COM objects and accessing its methods and properties.

5.3.2 Using the SIDEXIS XG PlugIn Wizard for Visual C++

This is by far the the easiest way to create your PlugIn. A helpful step-by-step introduction is given in the next chapter. The wizard incorporates a rapid application programming style offering the following features:

- Generation of a complete class framework, ready to compile immediately
- Configurable selection of custom filter parameters
- Sample filter implementation and filter dialog, can be modified immediately
- Most COM programming aspects are totally hidden, only MFC programming knowledge required

Using the SIDEXIS XG PlugIn Wizard is recommended if your PlugIn project can be realized with the Visual C++ environment and your requirements deal with standard filter applications.

This tool is a special purpose application wizard for Visual C/C++ integration. It is strongly recommended to start new SIDEXIS PlugIn projects using this provided wizard.

5.4 PlugIn Wizard Installation

Simply copy the following files from the SDK software distribution to your local Visual C/C++ template directory before you create a new PlugIn project. The needed files are contained in the SDK.ZIP distribution inside the \XGExeWizard directory:

Source:	\XGExeWizard.awx
	\XGExeWizard.hlp
Target:	\Microsoft Visual Studio\Common\MSDev98\Template\

5.5 Using the PlugIn wizard – a sample walkthrough

Using the PlugIn wizard is very easy. It takes only a simple 3-step construction workflow to finish the first compilable version of your new custom Plugin.

5.5.1 Creating a new project

For the creation please use the New option inside Microsoft Visual Studio. This will show up all the wizards installed on your system.

To set up a SIDEXIS PlugIn project you have to choose the "SIDEXIS XG Exe plug-in wizard". There you have to enter a name for your new project. This name can be different from the name of your PlugIn function name which will be entered on a page later on.

Please don't forget to specify the location where the PlugIn project should be generated.

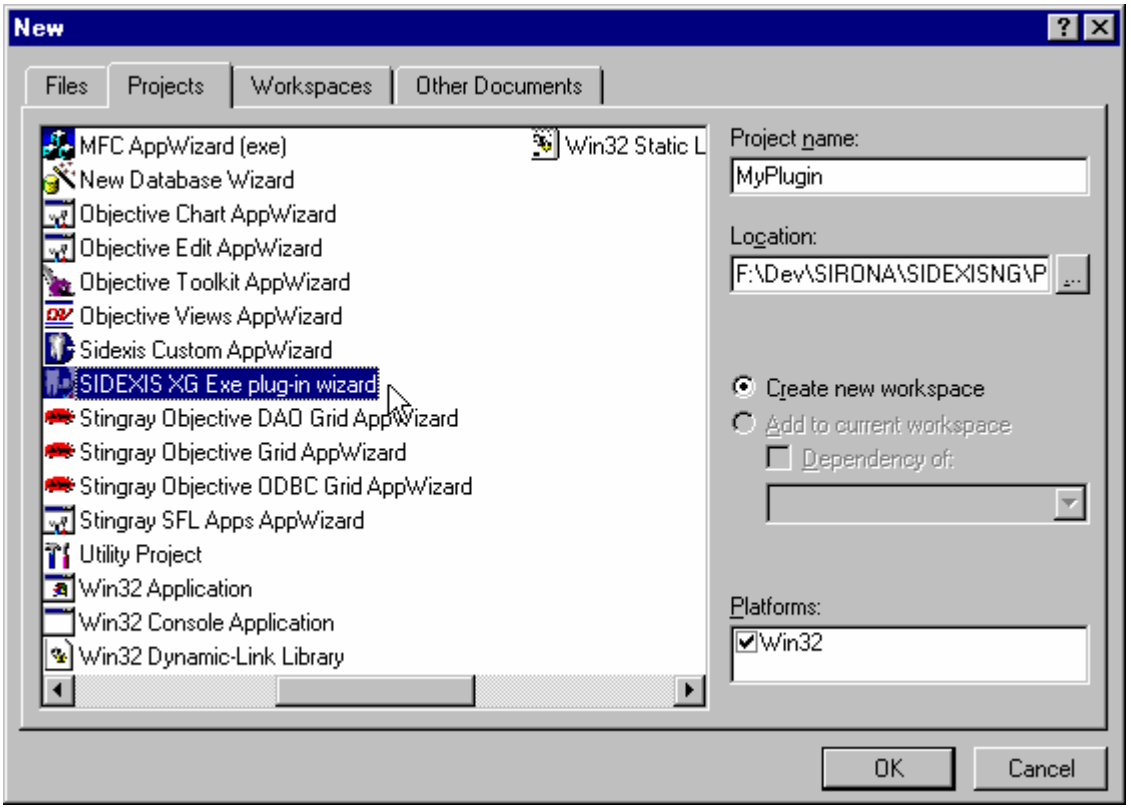

Figure 1 New Project

5.5.2 Common information

The first step for creating the new PlugIn project asks for information about yourself. This information will be used for creating a special project for you with predefined comment blocks customized with your name and company.

The version field represent the version of your PlugIn.

Figure 2 Wizard step 1, Common informations

5.5.3 PlugIn information

The second step requests information about the type of the PlugIn you wish to create and informations about the PlugIn.

The type you are creating depend on the functionality you wish to implement. If you want to create a PlugIn which can be started regardless of the state of SIDEXIS (opened image or not) you have to choose the "void plug-in" option. If your PlugIn can only be called if there is an already opened image you should use the option "filter plug-in".

The name for the PlugIn will be used to craete a class with the appropriate name including the ProgID.

The text you are entering in the description field will be shown as descriptive information inside SIDEXIS. You should use this field only in English language.

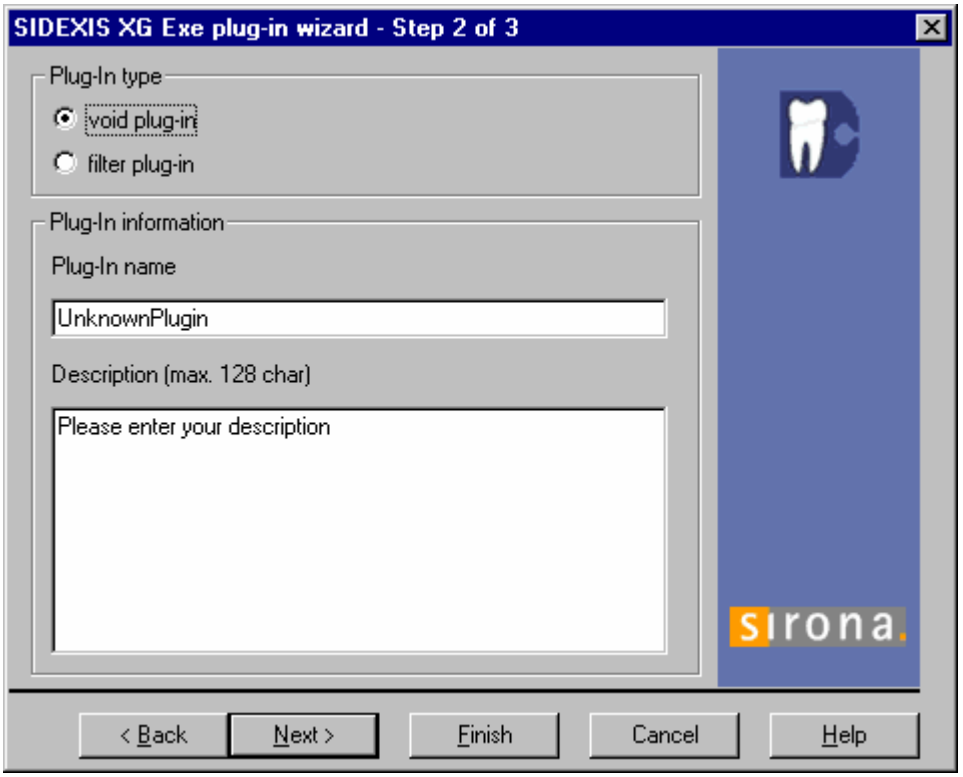

Figure 3 Wizard step 2, PlugIn informations

5.5.4 PlugIn GUI information

The third step allows you to enter the informations which will be used on the SIDEXIS user interface to identify your PlugIn. These informations will be shown on the PlugIn-manager page as well as on the GUI itself (buttons, tooltips, statusbars).

If your PlugIn has a need for parameters which the user has to enter you can checkmark the checkbox "Show parameter dialog" and the wizard will create a dialog for you to customize.

Figure 4 Wizard step 3, PlugIn GUI informations

5.5.5 Finishing the wizard

The last step shows up file creation information about the project to be created.

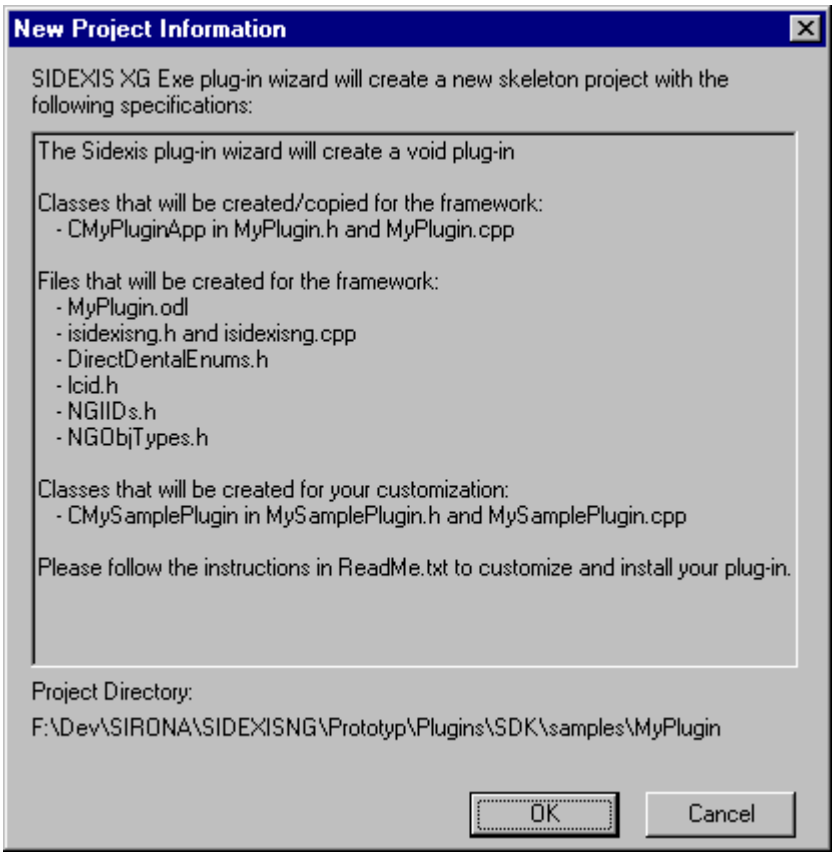

Figure 5 New project information

5.5.6 The project

Right after the creation you will see the project workspace. The wizard has created sample code for you so you can compile it and add it to the list of SIDEXIS PlugIns without any modification.

The project files represent the starting point for your custom modifications. Using the SIDEXIS programming model and the provided entry points you can add your PlugIn functionality. The wizard generated framework allows immediate integration and testing steps inside SIDEXIS.

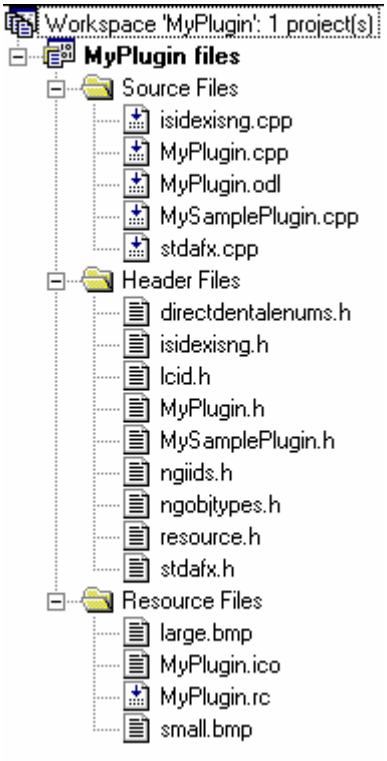

Figure 6 Files created

The PlugIn functionality has to be realized in the function "Run" from your PlugIn interface class. Add your custom code inside this method to alter the functionality of the created sample PlugIn.

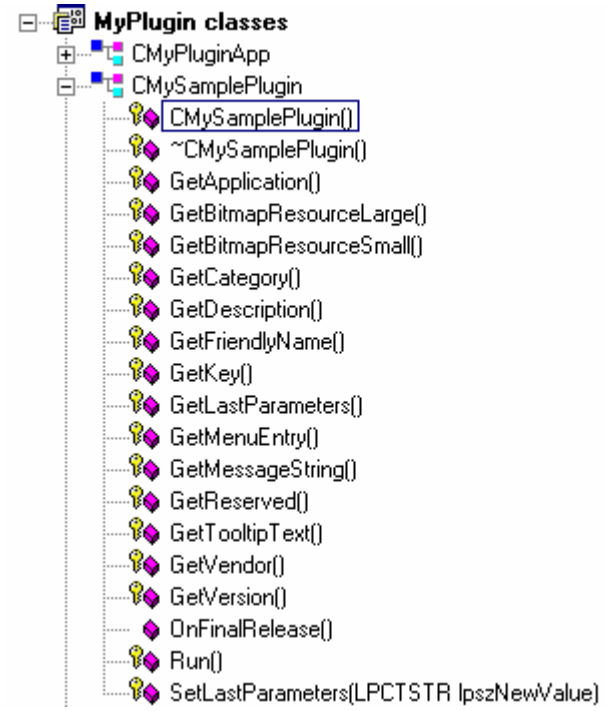

Figure 7 Central DirectDental interface class

5.6 SIDEXIS XG's COM Programming Model

SIDEXIS offers a collection of COM components which my be used by a custom plugin. To establish a bidirectional communication the plugin COM component has to keep a defined interface following the DirectDental standard.

The programming model covers most aspects of SIDEXIS XG's objects like examinations, images, drawing objetcs and patients.

For a detailled reference please refer to [2].

Some important top level interfaces:

Component/ Interface	**Description**
IDirectDental	This is the COM interface standard every PlugIn must follow to be compatible with SIDEXIS
IApplication	The programming model entry to get access to the SIDEXIS PM. It is the root of all other PM objects. The IApplication object of a running SIDEXIS instance can be connected through the "SIDEXISNG.Application" ProgID. Having this connection all other items of the PM can be accessed in a top-down strategy.
IExam	This PM object wraps (available) open exams
IImage	One of the most prominent exam memebers – the actual image data object

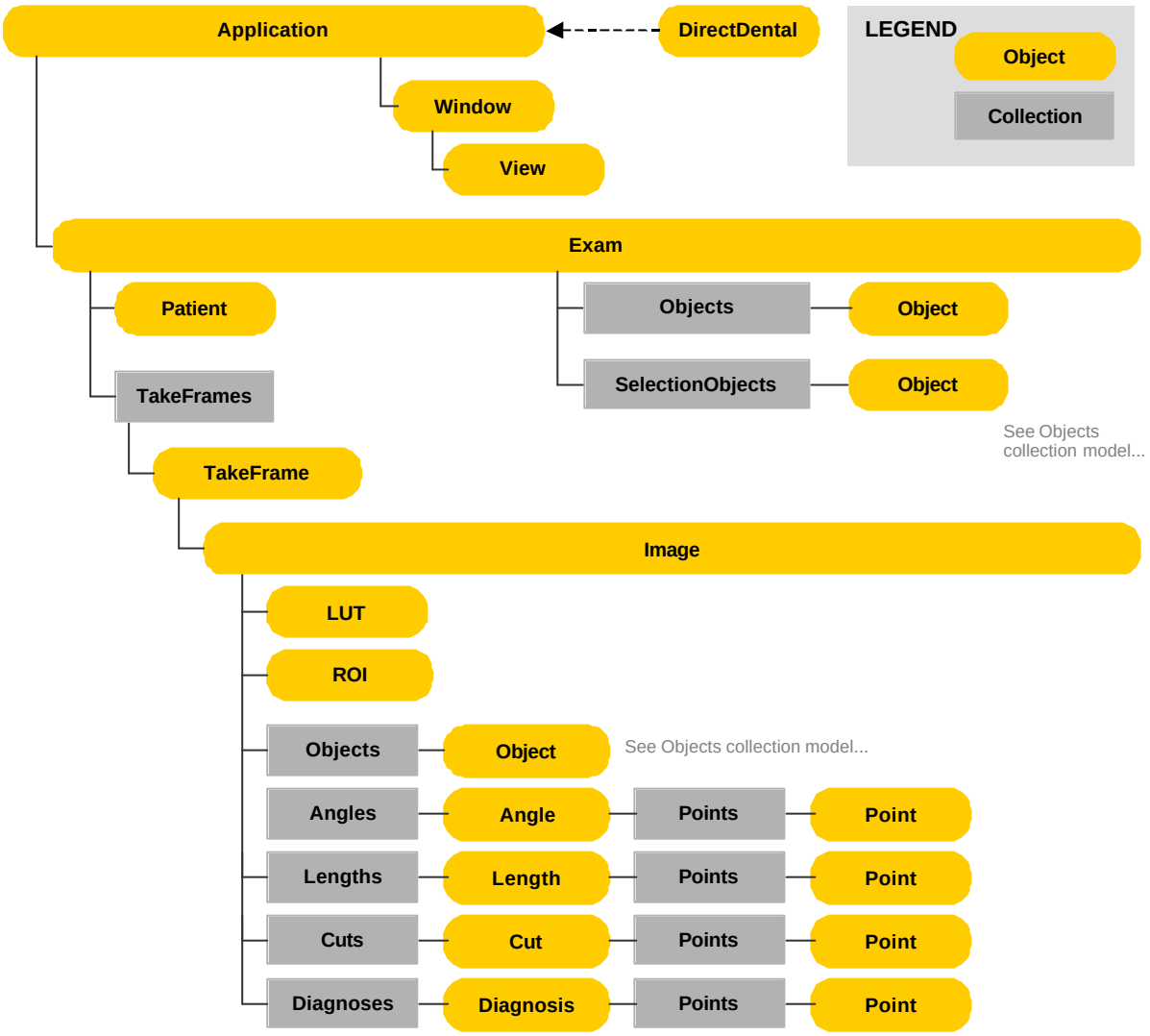

Figure 8 SIDEXIS programming model overview

5.7 Programming Model Concepts

Figure 9 presents an overview about visible SIDEXIS programming model objects. The yellow notes describe each object refering to it's corresponding programming model name. you can refer to [2] for a detailed description of each object's methods and properties.

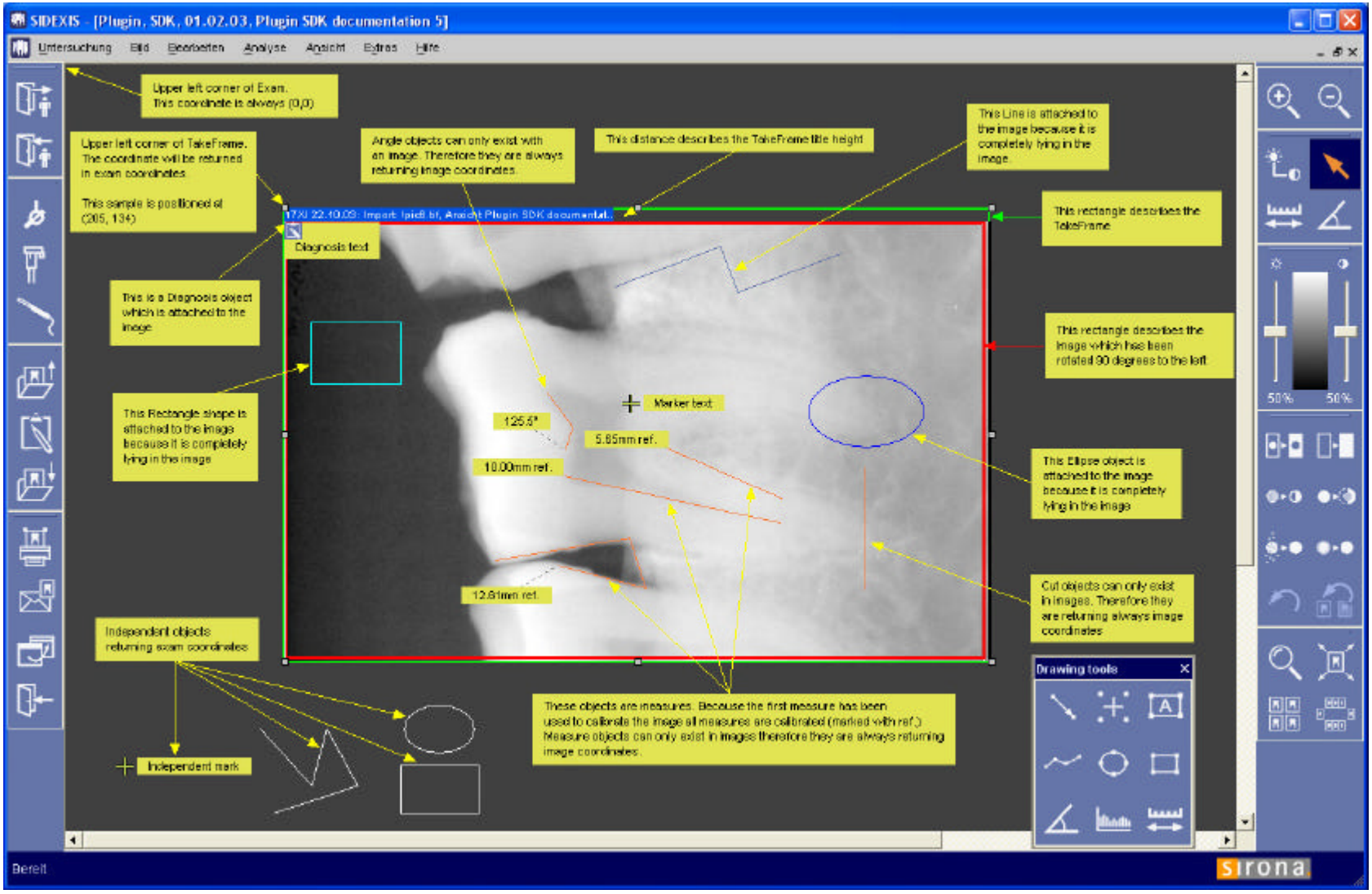

Figure 9 Visible PM objects

5.7.1 Coordinate systems

Pay attention to the mentioned coordinate systems. Starting on the top left you can find the *exam coordinate system* origin [0,0]. The position of all subsequent objects are returned in exam coordinates.

Looking inside an open Image object all corresponding subsequent objects are managed using the *image coordinate system*. The origin of this coordinate system is always the left top corner of the Image object.

Images may be rotated. For performance reasons, after a 90° rotation, Image.Width and Image.Height will be swapped. How to obtain rotation information and how to obtain the underlying coordinates see chapter "Length measurement and pixel values scenario".

Don't go out without this small code snippet on how to get the physical memory location of a pixel in a given Image rotation:

```
CPoint GetPixelMem( Iimage& Image, Cpoint& ptPixelImage ) // Pixel im Image coordinates
{
   CPoint ptRotated; // Return value in "memory" coordinates

   switch (Image.GetRotation())
   {
      case  90:
         ptRotated.x = ptPixelImage.y;
         ptRotated.y = (Image.GetHeight() - 1) - ptPixelImage.x;
         break;

      case 180:
         ptRotated.x = (Image.GetWidth()  - 1) - ptPixelImage.x;
         ptRotated.y = (Image.GetHeight() - 1) - ptPixelImage.y;
         break;

      case 270:
         ptRotated.x = (Image.GetWidth()  - 1) - ptPixelImage.y;
         ptRotated.y = ptPixelImage.x;
         break;

      case 0:
      default:
         ptRotated.x = ptPixelImage.x;
         ptRotated.y = ptPixelImage.y;
         break;
   }
   return ptRotated;
}
```

5.7.2 Typed objects

SIDEXIS allows two variants of accessing it's programming model objects:

1. Traversal of the specified object tree
2. Iteration of typed objects collections

Using the fist method the object types are given implicit by their position indside the object tree. The second method offers an iterative access through a pool of typed objects. The typed object itself may be accessed using the Object's TypedObject property.

The following figures illustrate both options:

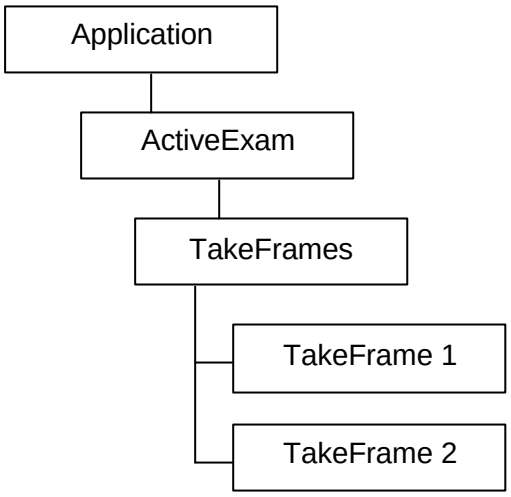

Figure 10 Hierarchical object tree

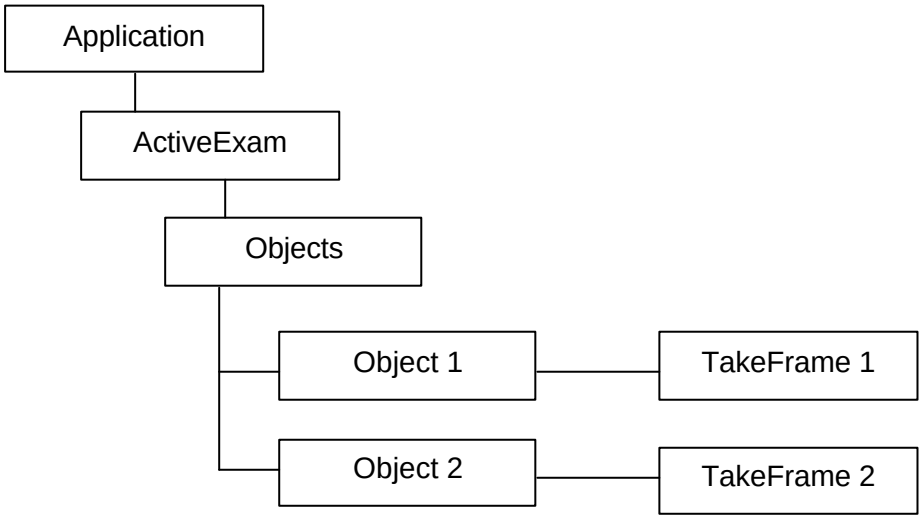

Figure 11 Collection of objects

Iterations of typed objects collections are ordered in descending overlay Z-order. This means f.ex. that in a list with TakeFrames, the active (= topmost) TakeFrame appears as the last item.

5.7.3 Length measurement and pixel values scenario

The starting point for this example is an open SIDEXIS exam and an embedded open image object. Inside this image some length measurements are placed manually. The scenario task is to retrieve the length measurement points and the corresponding greyscale pixel values at these places.

In the first section a valid exam object is retrieved. The exam object is the container of all embedded objects, including the expected length measurement objetcs.

Having an exam object we use the Objects collection to get access to the embedded subsequent objects.

```
IImage Image;

Image.AttachDispatch( pdispImage );

// Now iterating through the objects collection to look for the length objects
LPDISPATCH pdispObjects = Image.GetObjects();

if (!pdispObjects)
{
   AfxMessageBox( IDS_E_INTERNAL, MB_ICONSTOP | MB_OK );
   return;
}
IObjects Objects;

Objects.AttachDispatch( pdispObjects );

long lLengths = 0;
```

For each object we check its type. We're interested only in NGLength objects.

```
for (long lObjects = 0; lObjects < Objects.GetCount(); lObjects++)
{
   LPDISPATCH pdispObject = Objects.GetItem( lObjects );

   if (!pdispObject)
   {
      AfxMessageBox( IDS_E_INTERNAL, MB_ICONSTOP | MB_OK );
      return;
   }
   IObject Object;

   Object.AttachDispatch( pdispObject );

   switch (Object.GetType())
   {
```

Here we go. The following code snippet deals only with Length objects

```
      // Is it a length?
      case NGLength:
      {
         LPDISPATCH pdispLength = Object.GetTypedObject();

         if (!pdispLength)
         {
            AfxMessageBox( IDS_E_INTERNAL, MB_ICONSTOP | MB_OK );
            return;
         }
         ILength Length;

         Length.AttachDispatch( pdispLength );
```

After having retrieved the actual Length object as typed object through the object proxy we can access it's points collection.

```
         LPDISPATCH pdispPoints = Length.GetPoints();

         if (!pdispPoints)
         {
            AfxMessageBox( IDS_E_INTERNAL, MB_ICONSTOP | MB_OK );
            return;
         }
         IPoints Points;

         Points.AttachDispatch( pdispPoints );
```

Using the points collection we do iterate through all embedded single point objects and retrieve their position values in image coordinates.

```
         for (long lPoints = 0; lPoints < Points.GetCount(); lPoints++)
         {
            LPDISPATCH pdispPoint = Points.GetItem( lPoints );
```

```
if (!pdispPoint)
{
   AfxMessageBox( IDS_E_INTERNAL, MB_ICONSTOP | MB_OK );
   return;
}
IPoint Point;

Point.AttachDispatch( pdispPoint );
```

At this point we do retrieve the point coordinates of every single Point object. The following Point data can be used to set up a length calculation. In our case we retrieve the greyscale pixel value as additional information of the corresponding pixel. Since we want to obtain pixel *values* rather than coordinates we have to take into account the Image rotation in memory.

```
long lX = 0, lY = 0;

Point.Get( &lX, &lY );     // These are image coordinates

// Now get the pixel value for the corresponding point
CImage theImage;  // Pls. note: This wrapper only for greyscale images!

if (!theImage.FromSidexis( pdispImage ))
{
   AfxMessageBox( IDS_E_INTERNAL, MB_ICONSTOP | MB_OK );
   return;
}
HBITMAP  hBitmap = theImage.GetBitmap();
CBitmap* pBitmap = CBitmap::FromHandle( hBitmap );

CDC memDC;

memDC.CreateCompatibleDC( NULL );

CBitmap* pOld = memDC.SelectObject( pBitmap );
Cpoint   ptRotated;
int      nRotation = (int) Image.GetRotation();
Crect    rcRotated;

switch (nRotation)
{
   case  90:
      ptRotated.x = lY;
      ptRotated.y = (theImage.GetHeight() - 1) - lX;
      break;

   case 180:
      ptRotated.x = (theImage.GetWidth()  - 1) - lX;
      ptRotated.y = (theImage.GetHeight() - 1) - lY;
      break;

   case 270:
      ptRotated.x = (theImage.GetWidth()  - 1) - lY;
      ptRotated.y = lX;
      break;

   default:
      ptRotated.x = lX;
      ptRotated.y = lY;
      break;
}
COLORREF cr = memDC.GetPixel( ptRotated );
```

In this example all retrieved data is simply output to an edit window (see example below)

```
// output for this sample
CString strText;

strText.Format( IDS_OUTPUT, lLengths, lPoints, lX, lY,
                ptRotated.x, ptRotated.y,
                GetRValue( cr ), GetGValue( cr ), GetBValue( cr ) );
                pView->GetEditCtrl().ReplaceSel( strText, TRUE );
memDC.SelectObject( pOld );
}
```

```
            lLengths++;    // length number
            break; // case NGLength
         }
         default: break;
      }
   }
   if (!lLengths)
      AfxMessageBox( IDS_E_NOLENGTH, MB_ICONINFORMATION | MB_OK );
```

This sample will typically output the following text (Greyscale image):

```
Length[0] - Point[0] - Pixel(144, 190)      = 155 155 155
Length[0] - Point[1] - Pixel(602, 256)      = 255 255 255
Length[0] - Point[2] - Pixel(524, 498)      = 186 186 186
```

Please keep attention to the following issue: Length measurements inside image objects are visible to the user only if the starting and end point of the measurement objects are completely visible to the user. If any point corresponding to the measurement is "outside" the displayed TakeFrame (as shown in Figure 12) the length measurement note will not be displayed.

All length measurement objects and its properties can be retrieved in any case via programming model.

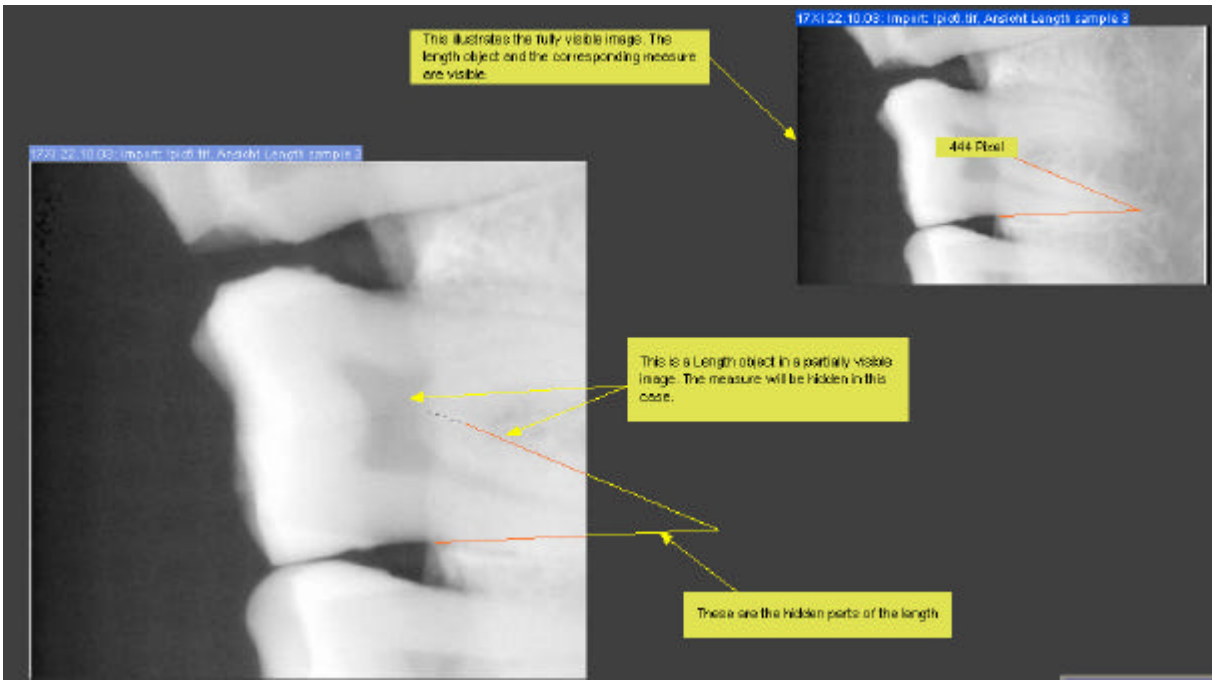

Figure 12 Hidden length measurements

5.7.4 Palette data retrieval scenario

Again the following code snippet starts with the active exam object aquisition. The task of this scenario is to extract and display palette data of an open image object inside this exam.

In the first section a valid exam object is retrieved. The exam object is the container of all embedded objects, including the expected image object.

Having an exam object we reference the active image object directly.

```
IExam Exam;

Exam.AttachDispatch( pdispExam );

// Retrieve the active image
LPDISPATCH pdispImage = Exam.GetActiveImage();
```

```
if (!pdispImage)
{
   AfxMessageBox( IDS_E_NOIMAGE, MB_ICONSTOP | MB_OK );
   return;
}
IImage Image;

Image.AttachDispatch( pdispImage );
```

Having an image object we can get access to it's embedded palette data

```
COleVariant varPalette;

SCODE sc = Image.GetPalette( &varPalette );

if (FAILED( sc ))
{
   AfxMessageBox( IDS_E_INTERNAL, MB_ICONSTOP | MB_OK );
   return;
}
```

The palette is handled through an OLE safearray and consists of a byte array of RGBQUADs.

```
COleSafeArray sa;
BYTE*         pPalette = NULL;  // The palette
long          lLBound      = 0;     // Upper Bound
long          lUBound      = 0;     // Lower Bound

sa.Attach( varPalette );
sa.GetLBound( 1, &lLBound );
sa.GetUBound( 1, &lUBound );

long lByteSize       = (lUBound - lLBound + 1);
long lPaletteEntries = lByteSize / sizeof( RGBQUAD );

sa.AccessData( (PVOID*) &pPalette );

RGBQUAD* pPaletteWork = new RGBQUAD[ lPaletteEntries ];

(void) memcpy( pPaletteWork, pPalette, lByteSize ); // [Bytes]

sa.UnaccessData();
```

Now having access to the complete palette array we report it to a local text object. A typical output will be shown below.

```
for (long lEntries = 0; lEntries < lPaletteEntries; lEntries++)
{
   CString strText;

   strText.Format( _T("Palette[%d] RGB 0x%02X%02X%02X\r\n"),
                   lEntries,
                   pPaletteWork[lEntries].rgbRed,
                   pPaletteWork[lEntries].rgbGreen,
                   pPaletteWork[lEntries].rgbBlue );
   pView->GetEditCtrl().ReplaceSel( strText, TRUE );
}
delete pPaletteWork;
```

The shown sample typically provides the following output::

```
Palette[0] RGB 0x000000
Palette[1] RGB 0x010101
Palette[2] RGB 0x020202
Palette[3] RGB 0x030303
Palette[4] RGB 0x040404
Palette[5] RGB 0x050505
Palette[6] RGB 0x060606
Palette[7] RGB 0x070707
Palette[8] RGB 0x080808
Palette[9] RGB 0x090909
...
```

5.8 Sample IDirectDental PlugIn Walkthrough

This chapter presents a brief description of the most important parts of the provided Binarize sample PlugIn. The shown code samples are not complete (please refer to the sample project files) but offers a good overview about the main coding patterns.

We take a look at the Binarize PlugIn which is part of the StandardFilters sample PlugIn. This is a PlugIn implemented with Microsoft Visual C++.

Most of the work is done using the provided SIDEXIS XG PlugIn Wizard for Microsoft Visual C++, but for academic purposes we jump into code right now to see some important implementation details.

The following walkthrough focuses on the usage of the IDirectDental COM interface provided by your custom PlugIn.

5.8.1 Cbinarize's PlugIn interface

Let's start with the IdirectDental-compatible interface. CCmdTarget is used as MFC COM-capable base class for easy COM interface generation. The generated ODL file is compatible to the IDirectDental standard. This interface will be called from SIDEXIS during runtime, especailly from the PlugIn manager residing inside SIDEXIS.

```
//---------------------------------------------------------------------------
/// Classname  : CBinarize : public CCmdTarget
///
/// This class is the plugin interface for the NG host.
///
//---------------------------------------------------------------------------
class CBinarize : public CCmdTarget
{
   //---------------------------------------------------------------------------
   // Member functions
   //---------------------------------------------------------------------------
public:
   DECLARE_DYNCREATE(CBinarize)
            CBinarize();
   virtual ~CBinarize();

   //{{AFX_VIRTUAL(CBinarize)
   public:
   virtual void OnFinalRelease();
   //}}AFX_VIRTUAL

protected:
   LPDISPATCH GetApplication();

   //{{AFX_MSG(CBinarize)
      // NOTE - the ClassWizard will add and remove member functions here.
   //}}AFX_MSG

   DECLARE_MESSAGE_MAP()
   DECLARE_OLECREATE_EX(CBinarize)

   //{{AFX_DISPATCH(CBinarize)
   afx_msg BSTR GetLastParameters();
   afx_msg void SetLastParameters(LPCTSTR lpszNewValue);
   afx_msg BSTR GetVendor();
   afx_msg BSTR GetDescription();
   afx_msg BSTR GetVersion();
   afx_msg BSTR GetCategory();
   afx_msg BSTR GetMenuEntry();
   afx_msg BSTR GetMessageString();
   afx_msg BSTR GetTooltipText();
   afx_msg BSTR GetFriendlyName();
   afx_msg BSTR GetBitmapResourceSmall();
   afx_msg BSTR GetBitmapResourceLarge();
   afx_msg BSTR GetKey();
```

```
   afx_msg SCODE Run();
   afx_msg BSTR GetReserved();
   //}}AFX_DISPATCH
   DECLARE_DISPATCH_MAP()

   DECLARE_INTERFACE_MAP()

   //--------------------------------------------------------------------------
   // Member variables                                    - Short description
   //--------------------------------------------------------------------------
protected:
   CCustomImage m_theImage;
};
```

5.8.2 GetFriendlyName: A sample Property read implementation

The following code fragment shows how to provide a language compatible fiendly name property value. First of all a instance of the SIDEXIS programming model root, the IApplication object, is retrieved.

Inside the \IDirectDental SDK directory you'll find pregenerated wrappers for the SIDEXIS PM. Just add them to your project and bind the required PM instances through valid LPDISPATCH interfaces (::GetApplication is a brief example for this step).

Using the bound IApplication the associated language ID can be read and used for custom language specific purposes.

```
BSTR CBinarize::GetFriendlyName()
{
   CString      strResult;
   IApplication Application;

   Application.AttachDispatch( GetApplication() );

   switch (Application.GetLanguage())
   {
      case LANGUAGE_ENGLISH_US:
         strResult = _T("Binarisation");
         break;

      case LANGUAGE_GERMAN:
         strResult = _T("Binarisierung");
         break;

      default:
         strResult = _T("Binarisation");
         break;
   }
   return strResult.AllocSysString();
}
```

In this sample a very useful helper, GetApplication(), is used and looks like the following implementation. This method should be available in nearly all PlugIns.

```
LPDISPATCH CBinarize::GetApplication()
{
   // Retrieve ClassID
   CLSID   clsid;
   HRESULT hr = CLSIDFromProgID( L"SIDEXISNG.Application", &clsid );

   if (FAILED( hr ))
      return NULL;

   // Find the active object
   LPUNKNOWN pUnkSrv  = NULL;

   hr = GetActiveObject( clsid, NULL, &pUnkSrv );  // returns Unknown pointer
   if (FAILED( hr ))
      return NULL;
```

```
   // Get the Dispatch interface
   LPDISPATCH pdispApp = NULL;

   hr = pUnkSrv->QueryInterface( IID_IDispatch, (void**) &pdispApp ); // returns IDispatch
   (void) pUnkSrv->Release(); // We don't need this anymore
   if (FAILED( hr ))
      return NULL;

   return pdispApp;
}
```

5.8.3 Run: The PlugIn activation

The one and only required IDirectDental method that actually activates the PlugIn.

During the beginning, an IApplication instance is bound and used to get access to the SIDEXIS programming model.

Using Application.GetActiveExam() the PlugIn first checks if there is any open exam available. Only open exams can host any image data which is checked right afterwards using Exam.GetActiveImage(). Note: The IExam object is part of the already bound IApplication.

Having got access to a valid Image, the image data is accessed through a local image wrapper, m_theImage. This helper object is responsible for the initial parameter dialog display and for the consecutive processing of the image data (for a detailed review of this wrapper please refer to the sample code)

After manipulation of the single pixels, the whole dataset is retransferred to SIDEXIS.

```
SCODE CBinarize::Run()
{
   IApplication Application;

   Application.AttachDispatch( GetApplication() );

   LPDISPATCH pdispExam = Application.GetActiveExam();

   if (!pdispExam)
   {
      OutputDebugString( _T("CBinarize::Run() No active examination\n") );
      return S_OK;
   }
   IExam Exam;

   Exam.AttachDispatch( pdispExam );

   LPDISPATCH pdispImage = Exam.GetActiveImage();

   if (!pdispImage)
   {
      OutputDebugString( _T("CBinarize::Run() No active image\n") );
      return S_OK;
   }
   if (!m_theImage.FromSidexis( pdispImage ))
   {
      OutputDebugString( _T("CBinarize::Run() Couldn't get image pixel map\n") );
      return S_OK;
   }
   // Do the work!
   if (FAILED( m_theImage.GetBinarizeParameters() ))
   {
      OutputDebugString( _T("CBinarize::Run() Couldn't retrieve parameters\n") );
      return S_OK;
   }
   if (!m_theImage.Binarize())
   {
      OutputDebugString( _T("CBinarize::Run() Couldn't filter image pixel map\n") );
      return S_OK;
   }
   (void) m_theImage.ToSidexis( pdispImage );
   (void) pdispImage->Release();
```

```
   Exam.Invalidate();
   return S_OK;
}
```

5.8.4 Toolbar buttons: The PlugIn Resources

Displaying custom bitmaps on associated SIDEXIS toolbars, two bitmaps are required in the PlugIn's resource, one for each possible toolbar resolution. Both bitmaps are identified through its resource names which are returned using the GetBitmapResourceSmall() and GetBitmapResourceLarge() Methods:

```
BSTR CBinarize::GetBitmapResourceSmall()
{
   CString strResult = _T("IDB_BINARIZE_SMALL");

   return strResult.AllocSysString();
}

BSTR CBinarize::GetBitmapResourceLarge()
{
   CString strResult = _T("IDB_BINARIZE_LARGE");

   return strResult.AllocSysString();
}
```

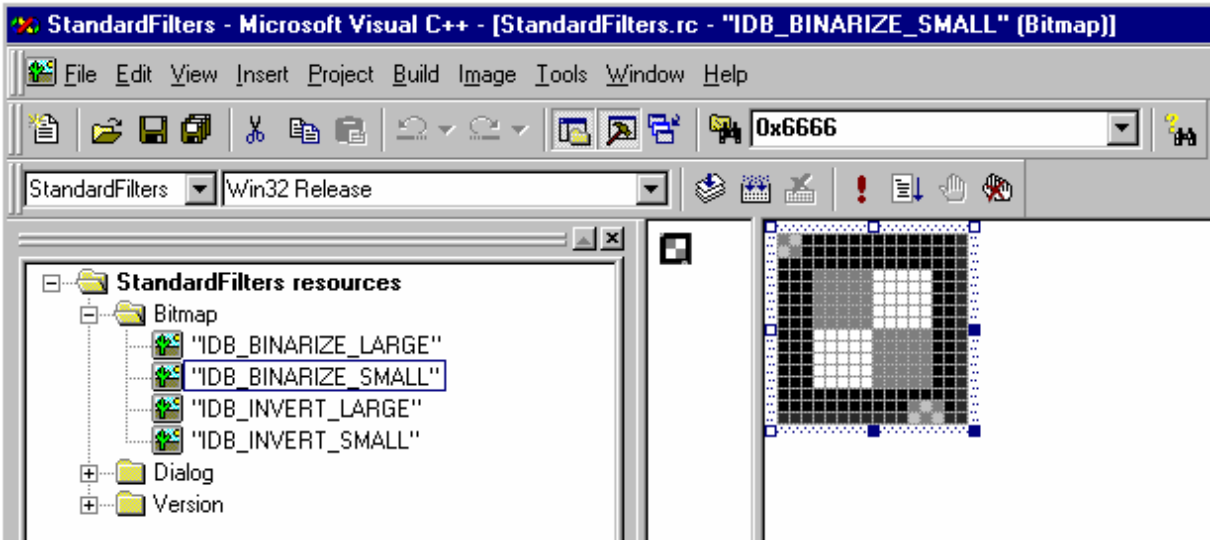

Figure 13 PlugIn bitmap resources

5.9 PlugIn COM registration

PlugIns must support proper COM registration and unregistration.

5.9.1 Dll registration

Dlls are registered like any COM DLLs through exporting the following functions:

DllRegisterServer()

DllUnregisterServer()

Both functions must be implemented.

5.9.2 Exe registration

To realize both registration and unregistration of custom EXE PlugIns, the recommended COM mechanisms for out-of-process COM servers should be incorporated (see MSDN documentation). This includes the usage of /UnregServer and /RegServer command line parameters.

The PlugIn application's startup code is responsible for the proper handling of both actions. An example code fragment:

```
TCHAR   szTokens[] = _T("-/");
LPCTSTR lpszToken = FindOneOf( m_lpCmdLine, szTokens );

while (lpszToken != NULL)
{
   if (lstrcmpi(lpszToken, _T("UnregServer"))==0)
   {
      COleObjectFactory::UpdateRegistryAll( FALSE ); // This unregisters all other COM
               objects derived from CcmdTarget
      return TRUE;
   }
   if (lstrcmpi(lpszToken, _T("RegServer"))==0)
   {
      if (COleObjectFactory::UpdateRegistryAll( TRUE ))
         OutputDebugString( _T("Successfully registered server.\n") );
      else
         OutputDebugString( _T("Warning: Server registration failed.\n") );

      return TRUE;
   }
   lpszToken = FindOneOf(lpszToken, szTokens);
}
```

5.9.3 Registering Plugins & OS access rights

Before you can use a Plugin with SIDEXIS it has to be registered with SIDEXIS' PlugIn Manager. Internally, the registration process includes 2 steps:

1. COM registration
2. SIDEXIS registration

For registering a COM server you ought to have writing privileges to the registry (HKEY_CLASSES_ROOT + HKEY_LOCAL_MACHINE). Remember that you don't have these writing privileges when logged in as a "normal", or, "main" user. Please perform the registration when you are logged in as (local) administrator!

In order to achieve this it is a good idea to bid the user call SIDEXIS' Plugin Manager while still logged in during your Plugin's custom setup. This prevents the "normal" user from getting poor Plugin Manager "Add Plugin" results on account of inappropriate access rights during her/his normal session.

5.10 Installing SIDEXIS PlugIns

SIDEXIS PlugIns must be configuerd both as new COM component inside the Windows system and as new SIDEXIS PlugIn inside the SIDEXIS configuration.

The following steps are required to get a new PlugIn installed:

1. Custom PlugIn setup. This step must be provided by the PlugIn provider and includes copying of all relevant files and objects/services to the target PC.
2. PlugIn registration using SIDEXIS' PlugIn Manager. This step registers the new PlugIn inside the SIDEXIS environment.

3. If SIDEXIS is installed in a network environment above steps must be repeated on every SIDEXIS workstation. For this purpose the PlugIn installation from step 1 can be placed on a shared network volume accessible from all workstations. The registration itself must be repeated on every workstation!

5.10.1 Updates

Of course, PlugIns can, and shall be, updated. After the PlugIn files themselves have been updated, return to SIDEXIS' PlugIn Manager to "Add" the updated PlugIn file. Via the PlugIns GUID, SIDEXIS automatically recognizes the case of a PlugIn already installed and treats the PlugIn as an update of the former one. The complete PlugIn information shown in the PlugIn Manager's details pane can be updated as well as the PlugIn's executable path.

The only information being not updateable are a PlugIn's category and symbol. In case you want to achieve this, you can work around that by simply first removing the PlugIn from the PlugIn Manager and re-add it again afterwards. The category is decisive for the position where the PlugIn is integrated into the SIDEXIS GUI. Thus, changing the category on the fly could cause conflicts.

Never change a PlugIn's GUID and category and keep in mind to maintain backward compatibility for filter PlugIns!

5.11 Debugging SIDEXIS PlugIns

5.11.1 Project settings

If you are debugging a PlugIn application, you should set the Visual Studio "**Program Arguments"** line of the "**Debug Options"** dialog to /Embedding or /Automation, so the debugger can launch the server application as though it were launched from a SIDEXIS container. Starting SIDEXIS from Program Manager or File Manager will then cause the container to use the instance of the server started in the debugger.

Now you can walk through your PlugIn code and set breakpoints as usual.

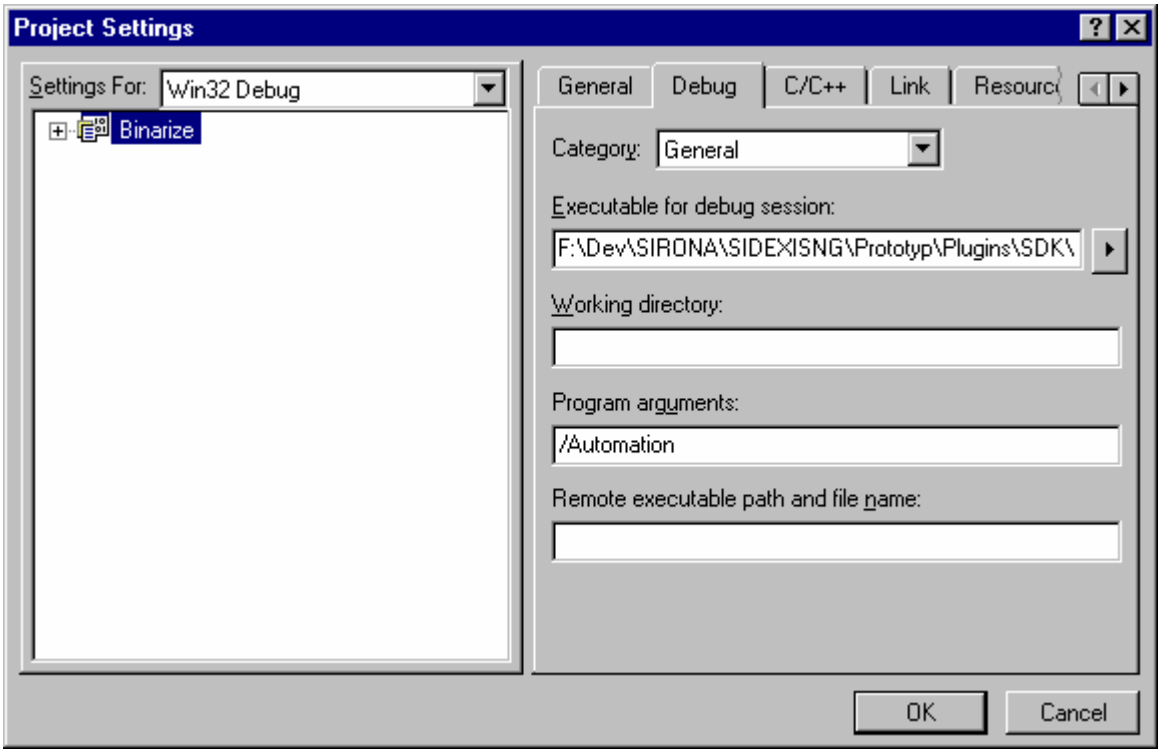

Figure 14 Debugging project settings

5.11.2 How can I debug a PlugIn?

The general mechanism of debugging COM components applies to the debugging of an IDirectDental PlugIn as well. Set the command line parameter to "/AUTOMATION" and start the debug command from the debugging IDE. The PlugIn will begin to execute and will wait for an instanciation of itself to be created. Now set a breakpoint at the beginning of your Run() method and execute the PlugIn from within SIDEXIS. By executing your PlugIn from SIDEXIS, this app will create an instance of the PlugIn's COM component and will call the Run() method. Now your debugger will get control over execution. Step through the PlugIn code for debugging purposes as usual.

6 Support

In case of developer support questions please e-mail to

develop.sidexis@sirona.de

Visit

www.SIDEXIS.com

in any case of question. There you'll find up-to-date background material, all documentations and the downloadable SDK.

Become a member of the ***SIDEXIS Developer Network*** community by registering your Plugin!

6.1 Frequently asked questions

Q I get an error message when trying to add a PlugIn in SIDEXIS XG bearing the text "Class not registered (...). Couldn't register PlugIn (...)". What went wrong?

A Your current login user account does not have enough access rights to add the PlugIn on your computer. Changing the registry must be enabled to allow COM registration. Please ask your administrator to adapt your logins's access rights accordingly.

Q Why does the debug mode finish automatically after return of the Run() method?

A Since the newly created instance will be released automatically by SIDEXIS after the Run() method returns the debugger will loose control over execution because the executed instance will be destroyed. Therefore, in order to obtain a break at the last breakpoint again, the PlugIn must be restarted in debug mode in the IDE as before.

7 Appendix

7.1 SDK contents

\Resources	Sample Plugin bitmaps Sample bitmap color palettes
\XGExeWizard	All required files for the PlugIn wizard
\Documentation	SDK documentation: ProgrammersGuide.pdf ProgrammersReference.pdf
\Samples\VC6\SDI[*)]	VC6 single document Plugin samples including source code
\Samples\VC6\MDI[*)]	VC6 multiple document Plugin samples including source code
\Samples\VC6\Dialog	VC6 dialogue based Plugin samples including source code

[*)] The SDI and MDI samples are very well suited for testing newly developped filter code: The samples are thus organized that "GetImage" can be done and the image will be displayed immediately inside the SDI/MDI view child window. "FilterImage" filters according to the pre-developped frame methods. That means that you would run through the same code as the PlugIn as if it were finished already and get displayed the filtered image. With "SetImage" you can send the filtered image pixels to SIDEXIS in order to see the result immediately. Very impressing here is that you can use SIDEXIS' menu command "Undo filter" although you do not (yet) have a real, finished PlugIn. So, you can develop & test very quickly.

7.2 SIDEXIS XG PlugIn SDK License Agreement

This is a legal agreement between you (either an individual or a single entity) and Sirona Dental Systems GmbH (hereinafter called Sirona) for the SIDEXIS XG PlugIn SDK and associated media and printed materials (hereinafter collectively called SDK) By installing or otherwise using SDK, you accept all the terms and conditions of this agreement. The SDK is owned by Sirona and is protected by international copyright laws. By this license agreement, Sirona grants you a non-exclusive license to use SDK on the terms set forth below.

1. Grant of License

An SDK license can be used by more than one individual developer. You may store or install a copy of the SDK on a storage device, such as a network file server, used only to install or run the SDK over an internal network.

2. Proprietary rights, copyright notices

Except for the limited license granted herein, Sirona, and its suppliers, retains exclusive ownership of all proprietary rights (including all ownership rights, title, and interest) in and to the SDK. If you are using the interfaces provided with the SDK in your application software, the application must contain the following copyright notice in the "About Box", "Splash Screen", or similar locations:

"Portions Copyright (C) 2003-2004 Sirona Dental Systems GmbH. All rights reserved."

3. Limitations

You may not use, copy, or modify the SDK, in whole or part, except as expressly provided for in this agreement. You may not rent, lease or sublicense the SDK. You may not reverse engineer, decompile or disassemble SDK binary components.

4. Term

Your SDK license is effective upon installation of the SDK. You may terminate the license at any time by destroying the SDK together with all copies and development results using the interfaces dealt with in the SDK. Your SDK license will terminate automatically if you fail to comply with any term or condition of this agreement.

5. Limited warranty and liability

Sirona warrants that the media containing the SDK (if provided by Sirona) is free from defects in material and workmanship. This is a limited warranty and it is the only warranty made by Sirona. Sirona makes no other warranty, representation, or condition, express or implied, and expressly disclaims the implied warranties of merchantability, fitness for a particular purpose, and noninfringement of third party rights. The limited warranty is void if failure of the SDK has resulted from accident, abuse or misapplication. Under no circumstances and under no legal theory, tort, contract, or otherwise, shall Sirona or its suppliers or resellers be liable to you or any other person for any indirect, special, incidental, or consequental damages of any character, including damages for loss of business profits, business interruption, computer failure or malfunction, or any and all other commercial damages or losses. In no case will Sirona be liable for any damages, even if Sirona had been informed on the possibility of such damages, or for any claim by any other party.

6. In general

This agreement represents the complete agreement concerning this license between the parties and supersedes all prior agreements and representations between them. This agreement may be amended or changed only in writing executed by both parties. This agreement shall be governed by and construed under the laws of the Federal Republic of Germany. The application of the United Nations Convention of Contracts for the International Sale of Goods is expressly excluded.

8 Index