SIDEXIS neXt Generation

GBE

Programmer's Reference

SIDEXIS XG Programming Model Version 1.2

Programmer’s Reference Version 1.2 (Released)

History

Date	Revision	Name
21.03.03	Review Version 1.1	JüZi
15.03.04	Review Version 1.2	JüZi

Product History

Date	Product Version	Name
21.03.03	SIDEXIS XG Version 1.1	JüZi
08.04.04	SIDEXIS XG Version 1.2	JüZi

1 Contents

2 Detailed Contents

3 Figures

4 Legal

Sirona Dental Systems GmbH reserves the right to revise this document and to make changes to its content at any time, without obligation to notify any person or entity of such revisions and change. Moreover, Sirona Dental Systems GmbH shall not be liable for errors contained herein or consequential damages in connection with the furnishing, performance, or use of this document.

Patents, patent applications, trademarks, copyrights, or other intellectual property rights may cover subject matter in this document. Except as expressly provided in any written license agreement from Sirona Dental Systems GmbH, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

SIDEXIS is a registered trademark of Sirona Dental Systems GmbH

Windows is a registered trademark of Microsoft Corporation

Other product and company names mentioned herein may be the trademarks of their respective owners.

5 Getting Started

5.1 About this document

This reference manual contains detailed information about the SIDEXIS neXt Generation programming model. Refer to this documentation to get a detailed SIDEXIS programming overview.

5.2 References

[1]	SIDEXIS XG Programmer’s Guide
[2]	SIDEXIS XG Programmer’s Referencel (this document)
[3]	SIDEXIS XG SDK license agreement (License.txt)

5.3 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
PI	DirectDental PlugIn
MSDN	Microsoft Developer Network
MFC	Microsoft Foundation Classes

5.4 Documentation Roadmap

To help you to 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. Contains general documentation background and navigation information.
Introduction	Refer to this section to get an overview about the SIDEXIS XG programming model.
IDirectDental Interface	Every plugin must provide a COM interface described in this section.
Programming Model Reference	This chapter is a detailed description of all programming model objects and their usage.

For further information please refer to the following documentations:

Document name	Contents
SIDEXIS XG Programmers Guide [1]	Overall introduction into the field of SIDEXIS XG programming
SIDEXIS XG Programmers Reference [2]	This documentation
SIDEXIS XG SDK license agreement [3]	Refer to License.txt in the SDK distribution and to appendix 10.5

5.5 Reference chapter organization

In chapters 7 and 8, each object is explained in a separate section following this structure:

Section	***Content***
Introduction	*Right after the object headline you'll find a brief overview about the object including a tabular description of all of it's exposed methods and properties.*
Working with....	*This section shows some typical applications of the associated object. Mostly you'll find here some typical source code examples.*
Methods	*This is the detailed method reference section.*
Properties	*This is the detailed properties reference section.*

6 Introduction

6.1 Programming with SIDEXIS XG

SIDEXIS exposes it's functionality through a hierarchical system of objects and collections of objects called an object or programming model. When you understand how to reference objects in an application's object model, you can use the available objects and features to build your custom PlugIns.

On top of SIDEXIS XG's programming model there is the Application object. This is the central entry point for all custom PlugIn projects.

Each object in the programming model exposes two types of interfaces:

- Methods – perform actions on objects
- Properties – determine/change characteristics of an object

Collections are special purpose objects containing a set of related objects, like:

- Image frames
- Diagnoses
- ...

6.2 PlugIn categories

Each PlugIn belongs to a certain pre-defined category. The following categories are currently defined:

Category	Description
"Filter"	The PlugIn is an image processing filter that can handle any image
„Filter/X-ray"	The PlugIn is an image processing filter that can handle only greyscale images
"Filter/Video"	The PlugIn is an image processing filter that can handle only true color images
"Void"	The PlugIn can do anything but filter

"Filter" PlugIns are only active as long as an image is active inside an active SIDEXIS exam. SIDEXIS stores information about every "Filter" PlugIn executed on the current image together with other data relevant to this image's view.

Since they are of general purpose, "Void" PlugIns are always active. In contrast to "Filter" PlugIns, nformation about their execution is not attached to an image's view.

6.3 Programming Model Overview

The following figures show all relevant SIDEXIS programming model objects and relations:

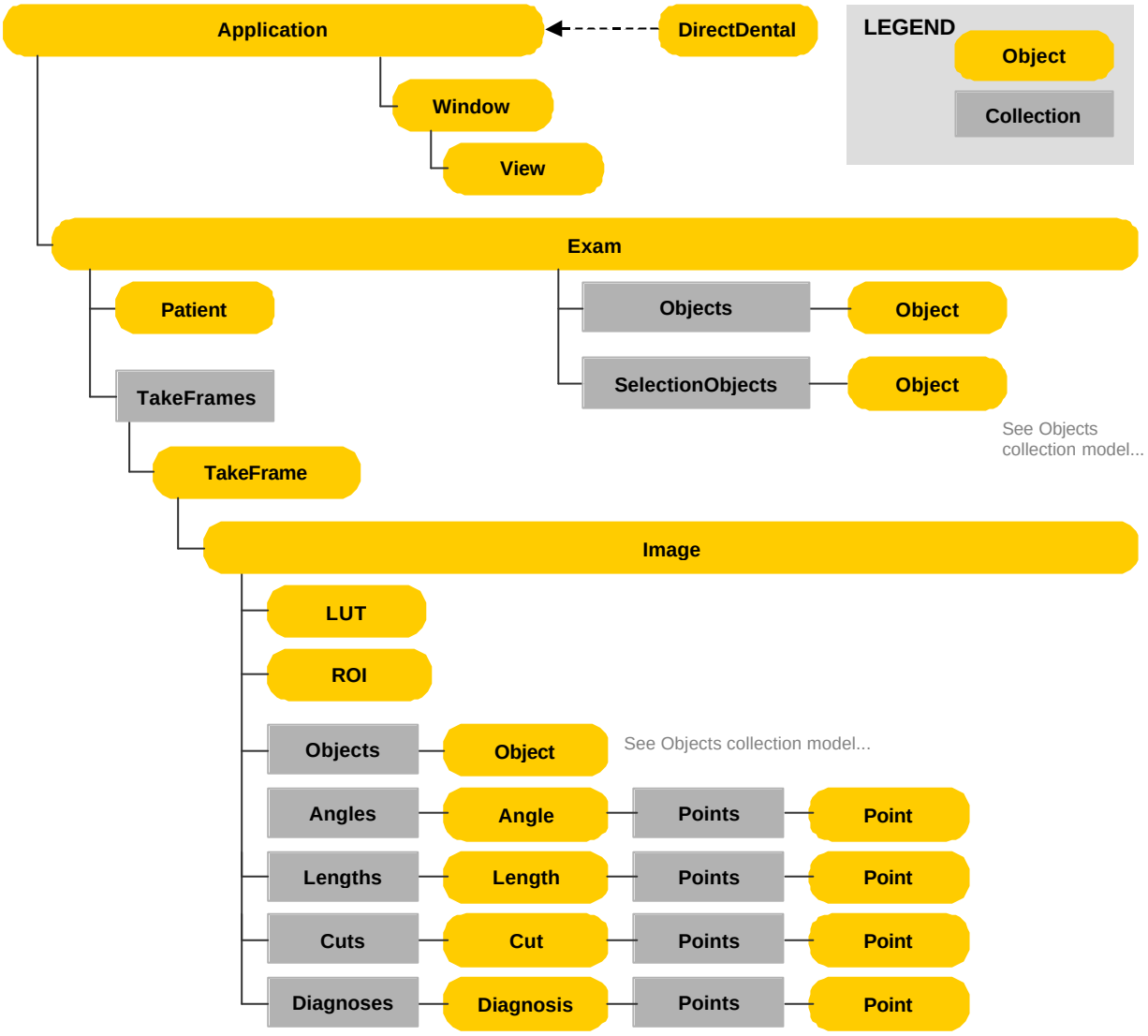

Figure 1 Object model overview

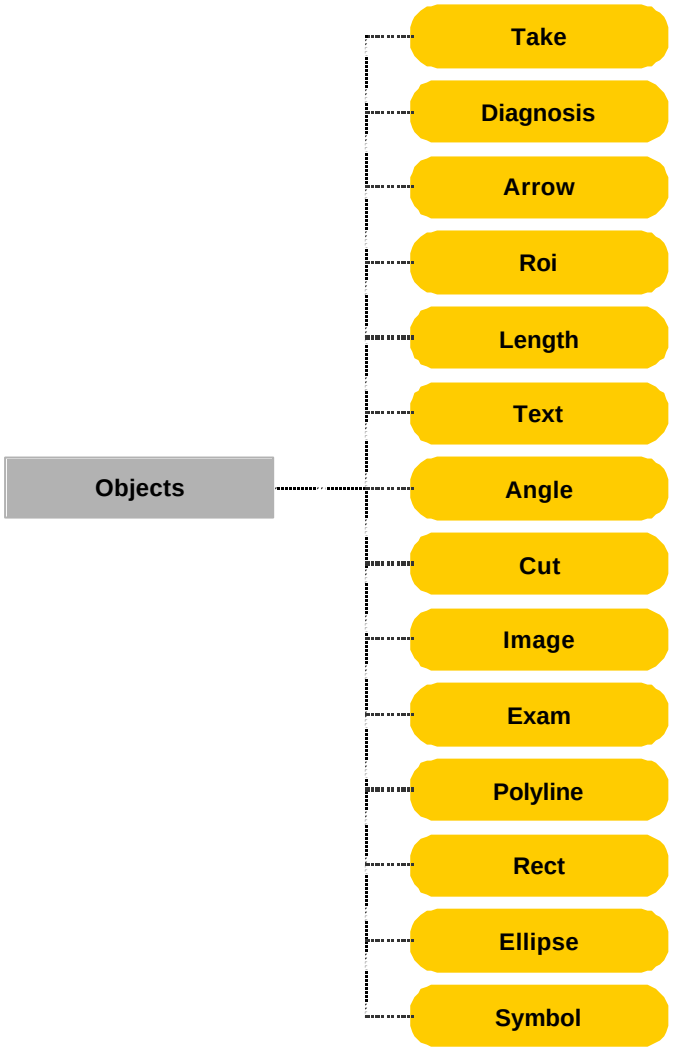

Figure 2 Objects collection and manageable object types

6.4 Coordinate systems

SIDEXIS uses three types of coordinate systems. There is the world coordinate system which is the same as your desktop, the Exam coordinate system and a coordinate system for each individual Image.

SIDEXIS includes several methods to transform coordinates from one system to another, e.g. the Exam object includes functions Screen2Exam and Exam2Screen which are able to transform to and from screen (desktop) coordinates to the Exam coordinate system.

The return type of the GetPosition functions depends on the status of the object you are requesting. Basically all objects which are attached to an image return their position in the Image coordinate system. This coordinate system may be rotated, f.ex. by using the image rotate functions of SIDEXIS. The rotation angle can be retrieved by using the Image function GetRotation.

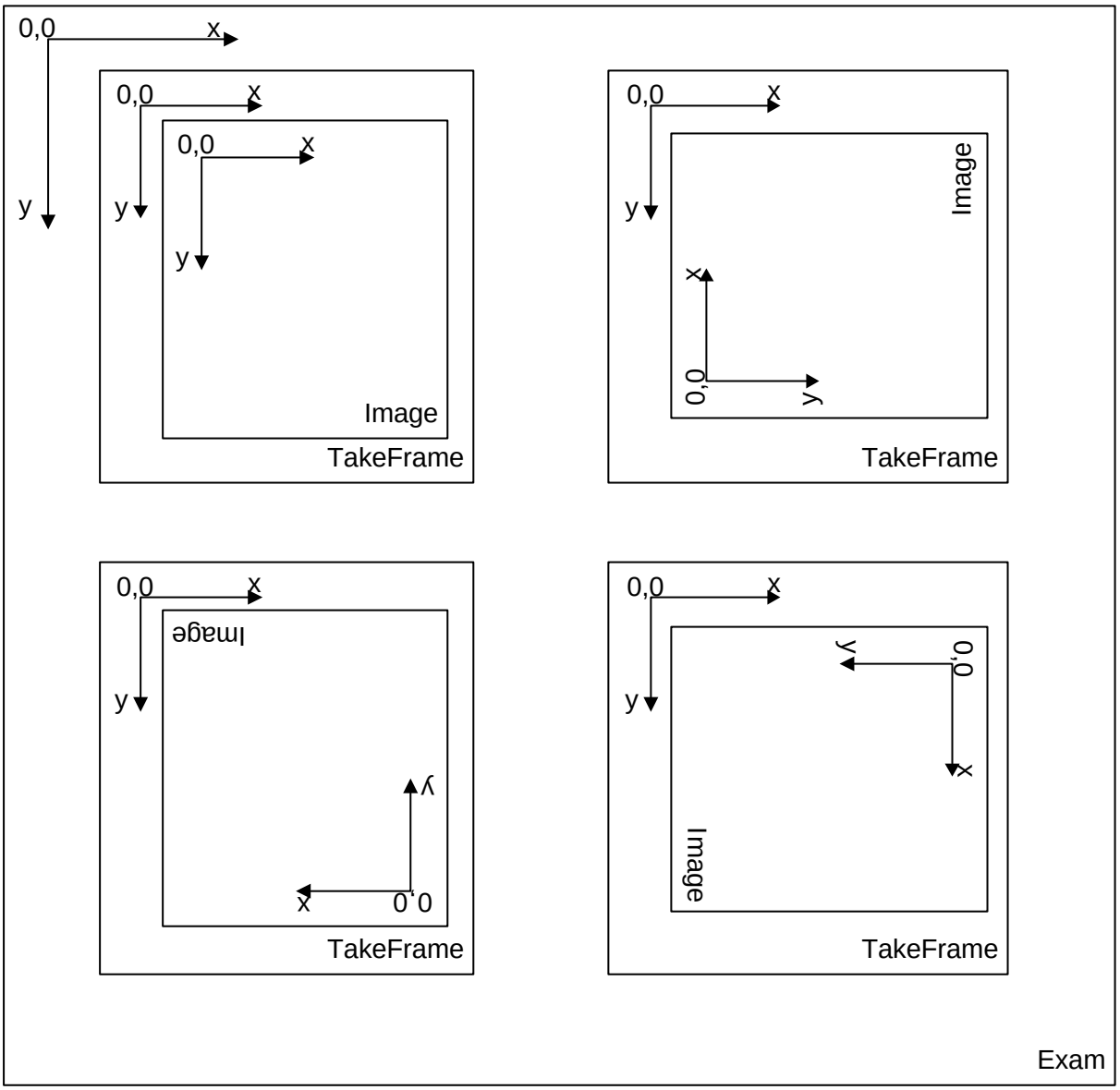

Figure 3 Coordinate systems overview

Please note that you should not have to worry about the rotation as long as you work inside the Image coordinate system. Coordinates of sub-objects inside this system like length measurements and so on have to be treated as if the Image were not rotated at all. Nevertheless rotation has to be taken into account in two cases:

1) When a pixel value at a given image coordinate is needed. For performance reasons, the image lies really physically rotated in memory. In order to obtain the correct pixel memory offset for a certain pixel you have to process pixel memory map coordinates from image coordinates according to the rotation. Especially, Image.Width and Image.Height are swapped in case of 90° and 270°!
2) When the absolute placement and direction of the image itself and its sub-objects inside the other coordinate systems is needed.

A detailed reference of coordinate system-dependent functions is embedded inside the individual object descriptions. For an example on how to work with rotated image coordinates see sample code in chapter “Working with the Image object”.

The unit of measure used in SIDEXIS is always pixel. The upper left corner of an exam is always (0; 0).

7 IDirectDental Interface

This is not an object provided by SIDEXIS, but rather an interface description that *your* PlugIn *must* implement. The SIDEXIS host is speaking to PlugIns using this interface standard.

Since IDirectDental uses Microsoft COM, the main PlugIn identification code is provided by its interface GUID. SIDEXIS will not be able to find and call a PlugIn any more with its GUID changed!

A custom PlugIn must provide the following methods:

Method	Return	Description
Run	SCODE	The one and only PlugIn activation method.

A custom PlugIn must provide the following properties:

Property	Type	Access	Description
LastParameters	BSTR	rw	This data is passed to a PlugIn just before the Run() method is triggered. After PlugIn termination any (new) data is stored inside SIDEXIS persistently and repassed to the PlugIn during the next activation.
Vendor	BSTR	r	Custom vendor string
Description	BSTR	r	PI description
Version	BSTR	r	PI version
Category	BSTR	r	PI category
MenuEntry	BSTR	r	PI specific menu entry
MessageString	BSTR	r	PI specific string for status window display purposes
TooltipText	BSTR	r	PI tooltip text
FriendlyName	BSTR	r	PI friendly name
BitmapResourceSmall	BSTR	r	Name of a bitmap file. This resource is placed on associated SIDEXIS toolbars (16x16 pixels).
BitmapResourceLarge	BSTR	r	Dito, large one (42x42 pixels)
Key	BSTR	r	Reserved, do not use this property
Reserved	BSTR	r	Reserved, do not use this property

7.1 Working with the IDirectDental Interface

We strongly recommend to use the “SIDEXIS XG PlugIn Wizard” to generate the DirectDental Interface object. Most of the work will be done by this wizard and you can concentrate on your custom modifications.

7.1.1 PlugIn identification

Since IdirectDental uses Microsoft COM, the main PlugIn identification method is provided by its interface GUID. SIDEXIS will not be able to find and call a PlugIn any more with its GUID changed!

7.1.2 PlugIn instantiation

The PlugIn host (SIDEXIS) instantiates COM components bearing the specially defined IDirectDental interface. At installation time (PlugIn Manager at work) the PlugIn is instantiated for the first time and the necessary information for setting up the PlugIn is determined by getting its propertes. In case of onrmal execution the COM interface is instantiated and the Run() method is called. After execution is complete the newly created instance will be released and therefore destroyed.

7.2 Methods

7.2.1 Run

The one and only PlugIn activation method.

Definition	SCODE Run()
Return	COM Errorcode
Params	-
C++ Usage	SCODE Run();
Comments	At this stage SIDEXIS passes control to the PlugIn. It is recommended to get a reference to the Application object for further programming model usage purposes. The PlugIn wizard automatically generates the following sample implementation: `SCODE CMySamplePlugin::Run()` `{` `  IApplication Application;` `  Application.AttachDispatch( GetApplication() );` `  AfxMessageBox( _T("SIDEXIS XG ") + Application.GetVersion(),` ` MB_ICONINFORMATION	MB_OK);` ` return S_OK;` `}`

7.3 Properties

7.3.1 LastParameters

In general, SIDEXIS saves original images. By applying filters, the original image can be altered into a different one that is called “view”. Those “views” can be saved persistently by the user for later automatic reproduction in other times or places, respectively. SIDEXIS queries for the parameters last recently used and keeps this information for later use, f.ex. in an undo history.

The data is passed to a PlugIn just before the Run() method is triggered in case of automatic revocation through SIDEXIS. After PlugIn execution (Run() method) the new/changed parameter set is

stored inside SIDEXIS persistently and repassed to the PlugIn during the next reload action of the "view".

Furthermore, this means that, in contrast to other functionality (Which can be used from other SIDEXIS Programming Model-aware applications like scripting engines and other executables directly without being a PlugIn), changing an image's "view" persistently is *only* possible via implementing a genuine IDirectDental filter PlugIn.

Type	BSTR
Access	Read, write
C++ Usage	BSTR GetLastParameters(); void SetLastParameters(LPCTSTR lpszNewValue);

Comments	This is the only possibility for a custom PlugIn to store persistent parameter data sets inside SIDEXIS. The string length must not exceed 1.024 characters! Typically used only in conjunction with filter PlugIns.
	The PlugIn wizard automatically generates the following sample implementation:
	`BSTR CMySamplePlugin::GetLastParameters()`
	`{`
	`  CString strResult = _T("");`
	`  return strResult.AllocSysString();`
	`}`

7.3.2 Vendor

Custom vendor string

Type	BSTR
Access	read
C++ Usage	BSTR GetVendor();

Comments	Return a custom vendor string using this property.
	The PlugIn wizard automatically generates the following sample implementation:
	`BSTR CMySamplePlugin::GetVendor()`
	`{`
	`  CString strResult = COMPANY;`
	`  return strResult.AllocSysString();`
	`}`

7.3.3 Description

This is your custom PlugIn description. It may contain multipile lines, separated by CR+LF ("\r\n").

Type	BSTR
Access	read
C++ Usage	BSTR GetDescription();

Comments	The PlugIn wizard automatically generates the following sample implementation:
	`BSTR CmySamplePlugin::GetDescription()`
	`{`
	`  CString strResult = DESCRIPTION;`
	`  return strResult.AllocSysString();`
	`}`

7.3.4 Version

Holds the custom PlugIn version.

Type	BSTR
Access	read
C++ Usage	BSTR GetVersion();

Comments	Of course, PlugIns can, and shall be in case of , updated. Please keep in mind that for filter PlugIn updates it is essential to kepp compatible with former versions in order to maintain past results. This means that every updated filter PlugIn must reckon with an automatic call from SIDEXIS revoking an ancestor version's command and parameter set! Currently, it is not possible - and not desired as to the user interface - to work with multiple instances different versions of the same PlugIn simultaneously.
	The PlugIn wizard automatically generates the following sample implementation:
	`BSTR CMySamplePlugin::GetVersion()`
	`{`
	`  CString strResult = VERSION;`
	`  return strResult.AllocSysString();`
	`}`

7.3.5 Category

This property defines your PlugIn's category.

Type	BSTR
Access	read
C++ Usage	BSTR GetCategory();

Comments	The category string must have one of the following values:
	`#define CATEGORY_VOID  _T("Void")`
	`#define CATEGORY_FILTER  _T("Filter")`
	"Void" Plugins can be activated anytime inside SIDEXIS. "Filter" Plugins require an already opened image object and are offered only in this case.
	The PlugIn wizard automatically generates the following sample implementation:
	`BSTR CMySamplePlugin::GetCategory()`
	`{`
	`  CString strResult = CATEGORY_VOID;`
	`  return strResult.AllocSysString();`
	`}`

7.3.6 MenuEntry

PlugIn-specific menu entry. It is inserted into the main SIDEXIS menu.

Type	BSTR
Access	read
C++ Usage	BSTR GetMenuEntry();

Comments	The PlugIn wizard automatically generates the following sample implementation:
	
	`BSTR CMySamplePlugin::GetMenuEntry()`
	`{`
	`  CString strResult = _T("MySamplePlugin");`
	
	`  return strResult.AllocSysString();`
	`}`

7.3.7 MessageString

PlugIn specific string for status window display purposes

Type	BSTR
Access	read
C++ Usage	BSTR GetMessageString();

Comments	The PlugIn wizard automatically generates the following sample implementation:
	
	`BSTR CMySamplePlugin::GetMessageString()`
	`{`
	`  CString strResult = _T("MySamplePlugin");`
	
	`  return strResult.AllocSysString();`
	`}`

7.3.8 TooltipText

PI tooltip text

Type	BSTR
Access	read
C++ Usage	BSTR GetTooltipText();

Comments	The PlugIn wizard automatically generates the following sample implementation:
	
	`BSTR CMySamplePlugin::GetTooltipText()`
	`{`
	`  CString strResult = _T("MySamplePlugin");`
	
	`  return strResult.AllocSysString();`
	`}`

7.3.9 FriendlyName

PlugIn friendly name. This gets displayed when SIDEXIS lists PlugIns or when a message concerning a certain PlugIn is brought up.

Type	BSTR
Access	read
C++ Usage	BSTR GetFriendlyName();

Comments	The PlugIn wizard automatically generates the following sample implementation:
	
	`BSTR CMySamplePlugin::GetFriendlyName()`
	`{`
	`  CString strResult = _T("MySamplePlugin");`
	
	`  return strResult.AllocSysString();`
	`}`

7.3.10 BitmapResourceSmall

The name of a bitmap resource. This resource is placed on associated small SIDEXIS toolbars (16x16 pixels).

Type	BSTR
Access	Read
C++ Usage	BSTR GetBitmapResourceSmall();

Comments	The PlugIn wizard automatically generates the following sample implementation:
	
	`BSTR CmySamplePlugin::GetBitmapResourceSmall()`
	`{`
	`  CString strResult = _T("PISmall.bmp");`
	
	`  return strResult.AllocSysString();`
	`}`
	
	For use when Application.GetVersion() returns “1.1”: The bitmap resource must be embedded into the executable.
	
	For use when Application.GetVersion() returns a value other than “1.1”: The bitmap resource can alternatively be delivered as a separate .bmp file. In this case, the return is the bitmap file name rather than a resource name. When the full path to the bitmap file is not given, SIDEXIS assumes the file to reside in the same directory as the PlugIn executable.
	
	A corresponding bitmap resource named "IDB_SMALL" will be generated, too.
	
	To use a bitmap for representing your PlugIn on the SIDEXIS GUI you should use the delivered grayscale palette in file “PISymbol.pal” or “PISymbol.act”. A sample bitmap file “PISmall.bmp” is included in the SDK distribution as a starting point for your custom designs, too.
	
	The palette colors are used by SIDEXIS for the representation of the PlugIn status as "grayed", "active" or "hovered". Because of this there is no need to deliver more than one picture (per size) for your PlugIn. The bitmaps generated by the PlugIn Wizard are already equipped with this palette.

7.3.11 BitmapResourceLarge

The name of a embedded bitmap resource. This resource is placed on associated large SIDEXIS toolbars (42x42 pixels)

Type	BSTR
Access	read
C++ Usage	BSTR GetBitmapResourceLarge();
Comments	Comments to BitmapResourceSmall apply here accordingly..

7.3.12 Key

Must be empty. Do not change the default implementation at the moment!

7.3.13 Reserved

Must be empty. Do not change the default implementation at the moment!

8 Programming Model Reference

This chapter describes all SIDEXIS programmable objects and interfaces.

8.1 Application Object

The Application object is the top level object and represents a running instance of SIDEXIS. Using this object, all subsequent objects can be created and manipulated.

Prog ID "SIDEXISNG.Application"

CLSID A5176796-B8FC-11D6-88BA-0050BF06B665

The following table describes the object's properties:

Property	Type	Access	Description
ActiveExam	object	r	Represents the active Exam Object
Visible	boolean	r	SIDEXIS visibility flag
Width	long	r	SIDEXIS application window width
Height	long	r	dito height
Version	BSTR	r	SIDEXIS application version
Path	BSTR	r	Path to SIDEXIS
Name	BSTR	r	SIDEXIS application name
FullName	BSTR	r	Path + name
StatusBar	BSTR	rw	Status bar string
ActivePatient	object	r	Active Patient object
ReadyState	long	r	Value of status enumeration
Language	long	r	Application's language locale ID

8.1.1 Working with the Application Object

The application object is the root of the XG object model. From there you can get all the information the model is exposing.

8.1.2 How to get the Application dispatch

```
LPDISPATCH CBinarize::GetApplication()
{
   HRESULT      hr = S_OK;
   CLSID        clsid;
   IApplication ISidexis;
   LPUNKNOWN    pUnkSrv  = NULL;
   LPDISPATCH   pdispApp = NULL;

   // Retrieve ClassID
   hr = CLSIDFromProgID( L"SIDEXISNG.Application", &clsid );
   if (FAILED( hr ))
      return NULL;

   // Find the active object
   hr = GetActiveObject( clsid, NULL, &pUnkSrv );  // returns Unknown pointer
```

```
        if (FAILED( hr ))
           return NULL;

        // Get the Dispatch interface
        hr = pUnkSrv->QueryInterface( IID_IDispatch, (void**) &pdispApp );   // returns
                   the dispatch Interface

        if (FAILED( hr ))
        {
           (void) pUnkSrv->Release();
           return NULL;
        }
        // We don't need this anymore
        (void) pUnkSrv->Release();

        return pdispApp;
     }
```

Once you have retrieved the application dispatch you can connect to it using the IApplication Interface. Simply do the following:

```
IApplication App;

App.AttachDispatch( pdispApp );

CString strVersion = App.GetVersion();
```

8.1.3 Properties

8.1.3.1 ActiveExam

Active examination object

Type	object
Access	read
C++ Usage	LPDISPATCH GetActiveExam();
Comments	See Exam object description for details.

8.1.3.2 Visible

Represents the visibility status of SIDEXIS

Type	boolean
Access	read
C++ Usage	BOOL GetVisible();
Comments	-

8.1.3.3 Width

Represents the width of the SIDEXIS main application window in pixels.

Type	long
Access	read
C++ Usage	long GetWidth();
Comments	-

8.1.3.4 Height

Represents the height of the SIDEXIS main application window in pixels

Type	long
Access	read
C++ Usage	long GetHeight();
Comments	-

8.1.3.5 Version

SIDEXIS XG's Programming Model version.

Type	BSTR
Access	read
C++ Usage	CString GetVersion();
Comments	-

8.1.3.6 Path

Path to the SIDEXIS.

Type	BSTR
Access	read
C++ Usage	Cstring GetPath();
Comments	-

8.1.3.7 Name

Name of the SIDEXIS application.

Type	BSTR
Access	read
C++ Usage	CString GetName();
Comments	-

8.1.3.8 FullName

Full name of the SIDEXIS application (Full path + file name).

Type	BSTR
Access	read
C++ Usage	CString GetFullName();
Comments	-

8.1.3.9 StatusBar

Status bar text display.

Type	BSTR
Access	read, write
C++ Usage	void SetStatusBar(LPCTSTR); CString GetStatusBar();
Comments	-

8.1.3.10 ActivePatient

Represents the Patient object of the active patient.

Type	object
Access	read
C++ Usage	LPDISPATCH GetActivePatient();
Comments	Returns an LPDISPATCH to Patient object of the active exam. Returns NULL when no exam is active. See Patient object reference for details.

8.1.3.11 ReadyState

Value of status enumeration

Type	long
Access	read
C++ Usage	long GetReadyState();
Comments	0 = Ready, 1 = not Ready

8.1.3.12 Language

Application's language locale ID

Type	Long
Access	Read
C++ Usage	long GetLanguage();
Comments	For a list of possible Ids, please refer to appendix.

8.2 Exam Object

An Exam objects represents a single (open) examination. It is the main container for all subsequent objects like images, annotations etc.

The following table describes the object's methods:

Method	Return	Description
GetItemCount	long	Returns the number of available components
Invalidate	void	Invalidates the whole examinatins to force a redraw
Screen2Exam	void	Returns Exam coordinates to given screen coordinates
IsSelected	boolean	Checks if a given object is selected

Exam2Screen	void	Returns screen coordinates to given Exam coordinates
InvalidateRect	void	Redraws all objetcs in a given area

The following table describes the object's properties:

Property	Type	Access	Description
ActiveImage	object	rw	Holds the active image object
Objects	object	r	The objects collection
Patient	object	r	The associated Patient object
TakeFrameTitleHeight	long	r	Title height in pixels
Name	BSTR	r	The examination's name
TimeStamp	DATE	r	It's time stamp
TakeFrames	object	r	The collection of available TakeFrame objects
ActiveTakeFrame	object	rw	The current active TakeFrame object
Height	long	r	The height of the examination area in pixels
Width	long	r	The width of the examination area in pixels
SelectionObjects	object	r	Collection of all currently selected Objects

8.2.1 Working with the Exam object

The active exam is the top level object for the current work in progress. It is related to the currently active patient. One possibly usage is to retrieve the currently selected objects.

```
IApplication app;

app.AttachDispatch( pdispApp );

LPDISPATCH pdispActiveExam = app.GetActiveExam();

if (!pdispActiveExam)
   return;                      // because there is no active exam

IExam Exam;

Exam.AttachDispatch( pdispActiveExam );

// retrieve the selected objects
LPDISPATCH pdispSelectionObjects = Exam.GetSelectionObjects();

if (pdispSelectionObjects)
{
   ISelectionObjects SelectionObjects;

   SelectionObjects.AttachDispatch( pdispSelectionObjects );

   for (long lSelectionObjects = 0; lSelectionObjects < SelectionObjects.GetCount();
lSelectionObjects++)
   {
      LPDISPATCH pdispObject = SelectionObjects.GetItem( lSelectionObjects );

      if (pdispObject)
      {
         IObject Object;
         Object.AttachDispatch( pdispObject );
      }
   }
}
```

8.2.1.1 The Exam coordinate system

The coordinate system of the Exam object is different from the world coordinate system (the desktop). The upper left corner is always (0, 0) and it's x axis is going from he left to the right and the y axis is going from top to bottom.

An objects which does not belong to an image, e.g. an angle or a cut, return their coordinates in this coordinate system.

If you are requesting the mouse position from the system using functions like GetCursorPos you will get screen coordinates. If you have to calculate the corresponding Exam coordinate you are able to use the Screen2Exam function.

8.2.2 Methods

8.2.2.1 GetItemCount

Returns the number of available components

Definition	long GetItemCount()
Return	number of components
Params	-
C++ Usage	long GetItemCount();
Comments	-

8.2.2.2 Invalidate

Invalidates the whole examination to force a redraw

Definition	void Invalidate()
Return	void
Params	-
C++ Usage	void Invalidate();
Comments	-

8.2.2.3 Screen2Exam

Returns Exam coordinates to given screen coordinates

Definition	void Screen2Exam(long lXScreen, long lYScreen, long* plXExam, long* plYExam)
Return	void
Params	lXScreen : x screen ordinate lYScreen : y screen ordinate plXExam : resulting x Exam ordinate plYExam : resulting y Exam ordinate
C++ Usage	void Screen2Exam(long lXScreen, long lYScreen, long* plXExam, long* plYExam);

Comments	See 8.2.1.1 for details concerning the SIDEXIS coordinate system.

8.2.2.4 IsSelected

Checks if a given object is selected

Definition	boolean IsSelected(IDispatch* pdispObject)	
Return	TRUE	: object is selected
	FALSE	: object is not selected
Params	pdispObject	: this object should be checked
C++ Usage	BOOL IsSelected(LPDISPATCH pdispObject);	
Comments	-	

8.2.2.5 Exam2Screen

Returns screen coordinates to given Exam coordinates

Definition	void Exam2Screen(	long lXExam, long lYExam, long* plXScreen, long* plYScreen)
Return	void	
Params	lXExam	: x Exam ordinate
	lYExam	: y Exam ordinate
	plXScreen	: resulting x screen ordinate
	plYScreen	: resulting y screen ordinate
C++ Usage	void Exam2Screen(long lXExam, long lYExam, long* plXScreen, long* plYScreen);	
Comments	See 8.2.1.1 for details concerning the SIDEXIS coordinate system.	

8.2.2.6 InvalidateRect

Redraws all objects in a given area

Definition	void InvalidateRect(	BOOL fUpdate, long nLeft, long nTop, long nWidth, long nHeight, IDispatch* pdispObject);	
Return	void		
Params	fUpdate	:	TRUE = redraw immediatly
	nLeft	:	left ordinate of redraw area [Image coordinate sys.]
	nTop	:	top ordinate of redraw area [Image coordinate sys.]
	nWidth	:	width ordinate of redraw area in pixels
	nHeight	:	height ordinate of redraw area in pixels

	pdispObject : alternative: invalidate this object only | NULL
C++ Usage	void InvalidateRect(long fUpdate, long nLeft, long nTop, long nWidth, long nHeight, LPDISPATCH pdispObject);
Comments	-

8.2.3 Properties

8.2.3.1 ActiveImage

Holds the active image object

Type	object
Access	read, write
C++ Usage	LPDISPATCH GetActiveImage(); void SetActiveImage(LPDISPATCH);
Comments	-

8.2.3.2 Objects

The objects collection

Type	object
Access	read
C++ Usage	LPDISPATCH GetObjects();
Comments	-

8.2.3.3 Patient

The associated Patient object

Type	object
Access	read
C++ Usage	LPDISPATCH GetPatient();
Comments	-

8.2.3.4 TakeFrameTitleHeight

Title height of TakeFrameTitle in pixels

Type	long
Access	read
C++ Usage	long GetTakeFrameTitleHeight();
Comments	Covers only the title area!

8.2.3.5 Name

The examination's name

Type	BSTR
Access	read
C++ Usage	CString GetName();
Comments	-

8.2.3.6 TimeStamp

The examination's time stamp

Type	DATE
Access	read
C++ Usage	DATE GetTimeStamp();
Comments	-

8.2.3.7 TakeFrames

The collection of the currently opened TakeFrame objects

Type	Object
Access	Read
C++ Usage	LPDISPATCH GetTakeFrames();
Comments	See 8.5 for TakeFrame object details. The TakeFrames in this list are ordered by descending overlay Zorder, i.e. from bottom to top. The active (= topmost) TakeFrame ist the last item in the list.

8.2.3.8 ActiveTakeFrame

The currently active TakeFrame object

Type	object
Access	read, write
C++ Usage	LPDISPATCH GetActiveTakeFrame(); void SetActiveTakeFrame(LPDISPATCH);
Comments	See 8.5 for TakeFrame object details

8.2.3.9 Height

The height of the examination area in pixels

Type	long
Access	read
C++ Usage	long GetHeight();
Comments	-

8.2.3.10 Width

The width of the examination area in pixels

Type	long
Access	read
C++ Usage	long GetWidth();
Comments	-

8.2.3.11 SelectionObjects

Collection of all currently selected Objects

Type	object
Access	read
C++ Usage	LPDISPATCH GetSelectionObjects();
Comments	-

8.3 Patient Object

An Exam may be associated with patient data. This information is accesible using the Patient object.

The following table describes the object's properties:

Property	Type	Access	Description
Lastname	BSTR	r	Patient's lastname
Firstname	BSTR	r	Patients' Firstname
DateOfBirth	DATE	r	Date of birth
ExternalID	BSTR	r	The external managed patient ID

The Patient object does not offer a method interface.

8.3.1 Working with the Patient object

The patient object relates 1:1 to the exam object.

```
LPDISPATCH pdispPatient = Exam.GetPatient();

if (pdispPatient)
{
   IPatient Patient;

   Patient.AttachDispatch( pdispPatient, FALSE );

   CString strLastname = Patient.GetLastname();
}
```

8.3.2 Properties

8.3.2.1 Lastname

Patient's last name

Type	BSTR
Access	read
C++ Usage	CString GetLastname();
Comments	-

8.3.2.2 Firstname

Patient's first name

Type	BSTR
Access	read
C++ Usage	CString GetFirstname();
Comments	-

8.3.2.3 DateOfBirth

Patient's date of birth

Type	DATE
Access	read
C++ Usage	DATE GetDateOfBirth();
Comments	-

8.3.2.4 ExternalID

The external managed patient ID

Type	BSTR
Access	read
C++ Usage	CString GetExternalID();
Comments	-

8.4 TakeFrames collection

Using the TakeFrames collection you can get access to all opened Take frames.

The following table describes the object's methods:

Method	Return	Description
Item	object	Returns an object by index

The following table describes the object's properties:

Property	Type	Access	Description
Count	long	r	Number of items in collection

8.4.1 Working with the TakeFrames collection

```
LPDISPATCH pdispTakeFrames = Exam.GetTakeFrames();

if (pdispTakeFrames)
{
   ITakeFrames TakeFrames;

   TakeFrames.AttachDispatch( pdispTakeFrames );

   // all Frames
   for (long lCount = 0; lCount < TakeFrames.GetCount(); lCount++ )
   {
      LPDISPATCH pdispTakeFrame = TakeFrames.GetItem( lCount );

      if (pdispTakeFrame)
      {
         ITakeFrame TakeFrame;

         TakeFrame.AttachDispatch( pdispTakeFrame, FALSE );
      }
   }
}
```

8.4.2 Methods

8.4.2.1 Item

Definition	Idispatch* Item(long lIndex);
Return	The requested object
Params	lIndex : index to item in collection
C++ Usage	LPDISPATCH GetItem(long lIndex);
Comments	-

8.4.3 Properties

8.4.3.1 Count

Number of items in collection

Type	long
Access	read
C++ Usage	long GetCount();
Comments	-

8.5 TakeFrame Object

The TakeFrame object is a frame for holding opened images and for images to be created.

The following table describes the object's methods:

Method	Return	Description
GetPosition	void	Returns frame position
SetPosition	void	Sets frame position
GetVisibleImagePosition	SCODE	Returns visible image part of a shown TakeFrame
SetVisibleImagePosition	SCODE	Sets visible image part of a shown TakeFrame
ZoomImage	SCODE	Image zooming
MoveTo	SCODE	Moves Frame to given position
RotateImage	SCODE	Image rotation

The following table describes the object's properties:

Property	Type	Access	Description
TitleHeight	long	r	TakeFrames title height
Type	BSTR	r	Type
Regio	BSTR	r	Regio
Name	BSTR	r	Name
Reason	BSTR	r	Image reason
Title	BSTR	r	Titlestring
Image	object	r	Associated Image object
Rotation	double	r	Applied rotation
ID	long	r	Component ID

8.5.1 Working with the TakeFrame object

```
LPDISPATCH pdispTakeFrames = Exam.GetTakeFrames();

if (pdispTakeFrames)
{
   ITakeFrames TakeFrames;

   TakeFrames.AttachDispatch( pdispTakeFrames );

   LPDISPATCH pdispTakeFrame = TakeFrames.GetItem( lCount );

   if (pdispTakeFrame)
   {
      ITakeFrame TakeFrame;

      TakeFrame.AttachDispatch( pdispTakeFrame, FALSE );

      LPDISPATCH pdispImage = TakeFrame.GetImage();

      if (pdispImage)
      {
         // Image
         IImage Image;

         Image.AttachDispatch( pdispImage, FALSE );
      }
      else
         ; // This is an empty TakeFrame
   }
}
```

8.5.2 Methods

8.5.2.1 GetPosition

Returns frame position in exam coordinates.

Definition	Void GetPosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);
Return	Void
Params	pnLeft : X position of top left corner [exam coordinates] pnTop : Y poition ot top left corner [exam coordinates] pnWidth : width in pixels pnHeight : height in pixels
C++ Usage	Void GetPosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);
Comments	-

8.5.2.2 SetPosition

Sets frame position in exam coordinates.

Definition	Void SetPosition(long nLeft, long nTop, long nWidth, long nHeight);
Return	Void
Params	nLeft : X position of top left corner [exam coordinates] nTop : Y poition ot top left corner [exam coordinates] nWidth : width in pixels nHeight : height in pixels
C++ Usage	Void SetPosition(long nLeft, long nTop, long nWidth, long nHeight);
Comments	-

8.5.2.3 GetVisibleImagePosition

Returns visible image part of a shown TakeFrame

Definition	SCODE GetVisibleImagePosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);
Return	COM errorcode
Params	pnLeft : X position of top left corner [exam coordinates] pnTop : Y poition ot top left corner [exam coordinates] pnWidth : width in pixels pnHeight : height in pixels
C++ Usage	SCODE GetVisibleImagePosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);
Comments	-

8.5.2.4 SetVisibleImagePosition

Returns visible image part of a shown TakeFrame

Definition	SCODE SetVisibleImagePosition(long nLeft, long nTop, long nWidth, long nHeight)
Return	COM errorcode
Params	nLeft : X position of top left corner [exam coordinates] nTop : Y poition ot top left corner [exam coordinates] nWidth : width in pixels nHeight : height in pixels
C++ Usage	SCODE SetVisibleImagePosition(long nLeft, long nTop, long nWidth, long nHeight);
Comments	-

8.5.2.5 ZoomImage

Image zooming method.

Definition	SCODE ZoomImage(long nZoomNumerator, long nZoomDenominator, BOOL fAdaptFrame);
Return	COM errorcode
Params	nZoomNumerator : zoom factor numerator nZoomDenominator : zoom factor denominator fAdaptFrame : must be TRUE
C++ Usage	SCODE ZoomImage(long nZoomNumerator, long nZoomDenominator, long fAdaptFrame);
Comments	-

8.5.2.6 MoveTo

Moves Frame to given position in exam coordinates.

Definition	SCODE MoveTo(long lX, long lY)
Return	COM errorcode
Params	lX : target X ordinate lY : target Y ordinate
C++ Usage	SCODE MoveTo(long lX, long lY);
Comments	-

8.5.2.7 RotateImage

Image rotation method.

Definition	SCODE RotateImage(double drDegrees)
Return	COM errorcode

Params	drDegrees rotation degress
C++ Usage	SCODE RotateImage(double drDegrees);
Comments	Only the following rotation degrees are valid: 0|90|180|270°

8.5.3 Properties

8.5.3.1 TitleHeight

Title height in pixels

Type	long
Access	read
C++ Usage	long GetTitleHeight();
Comments	-

8.5.3.2 Type

This property represents the take type of an order.

Type	BSTR
Access	read
C++ Usage	CString GetType();
Comments	Example: "XI"

8.5.3.3 Regio

This property holds the regio information.

Type	BSTR
Access	read
C++ Usage	CString GetRegio();
Comments	Example: "35", see 10.1 for details.

8.5.3.4 Name

Name property

Type	BSTR
Access	read
C++ Usage	CString GetName();
Comments	-

8.5.3.5 Reason

Reason/Notes for this image.

Type	BSTR

Access	read
C++ Usage	CString GetReason();
Comments	Reason/Notes for this images/these images

8.5.3.6 Title

Title property

Type	BSTR
Access	read
C++ Usage	CString GetTitle();
Comments	-

8.5.3.7 Image

This is the associated Image object

Type	object
Access	read
C++ Usage	LPDISPATCH GetImage();
Comments	See 8.9 for Image object details.

8.5.3.8 Rotation

Rotation degree property

Type	double
Access	read
C++ Usage	double GetRotation();
Comments	Only the following rotation degrees are valid: 0|90|180|270°

8.5.3.9 ID

ID property

Type	long
Access	read
C++ Usage	long GetId();
Comments	Reserved. Do not use this property.

8.6 Objects collection

The objects collection exists for various super-objects. For example the objects collection in the sample below the exam object holds all objects which belong directly to the exam object. On the other hand the objects collection below an Image object holds all objects which belong to this image object.

You can use the object collection for iterating all objects in a specific context without having to know which type the objects have.

The following table describes the object's methods:

Method	Return	Description
Item	object	Returns an object by index

The following table describes the object's properties:

Property	Type	Access	Description
Count	long	r	Number of items in collection

8.6.1 Working with the Objects collection

```
LPDISPATCH pdispObjects = Exam.GetObjects();

if (pdispObjects)
{
   IObjects Objects;

   Objects.AttachDispatch( pdispObjects );

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

      if (pdispObject)
      {
         IObject Object;

         Object.AttachDispatch( pdispObject );
      }
   }
}
```

8.6.2 Methods

8.6.2.1 Item

Definition	IDispatch* Item(long lIndex)
Return	The select object
Params	lIndex : index to item in collection
C++ Usage	LPDISPATCH GetItem(long lIndex);
Comments	This usage of this iteration method is similar in all SIDEXIS collection objects.

8.6.3 Properties

Number of items in collection

Type	long
Access	read
C++ Usage	long GetCount();
Comments	This usage of this property is similar in all SIDEXIS collection objects.

8.7 Object object

This object is used as a general object proxy for several objects in the SIDEXIS programming model.

The following table describes the object's properties:

Property	Type	Access	Description
Type	long	r	it's type
TypedObject	object	r	reference to the typed object

8.7.1 Working with the Object object

8.7.1.1 Code sample

The typical task for using the Object object is to retrieve the type for it and getting the right typed dispatch for that object for a typecast.

```
LPDISPATCH pdispObject = Objects.GetItem( lObjects );

if (pdispObject)
{
   IObject Object;

   Object.AttachDispatch( pdispObject );

   switch (Object.GetType())
   {
      case NGTake:
      {
          LPDISPATCH pdispTyped = Object.GetTypedObject();
         ITakeFrame TakeFrame;

         TakeFrame.AttachDispatch( pdispTyped );
         break;
      }
      case NGArrow:
      {
         LPDISPATCH pdispTyped = Object.GetTypedObject();
         IArrow Arrow;

         Arrow.AttachDispatch( pdispTyped );
         break;
      }
      default: break;
   }
}
```

8.7.2 Properties

8.7.2.1 Type

Holds the actual object type

Type	long
Access	read
C++ Usage	long GetType();

Comments	For a list of all possible object types (“NGNone” (=-1), “NGUnknown” (=0), “NGTake” (=1),...) please refer to appendix 10.2.

8.7.2.2 TypedObject

This is the actual object reference

Type	object
Access	read
C++ Usage	LPDISPATCH GetTypedObject();
Comments	This is the actual object accessor to get an instance of the corresponding typed object.

8.8 SelectionObjects collection

The SelectionObjects collection exists only for the exam object. This collection holds the Object dispatches for all selected objects. With this information you can iterate all selected objects and perform the task you want.

The SelectionObjects collection works similar to the ObjectsCollection. Please refer to 8.6 for details.

8.8.1 Working with the SelectionObject collection

```
LPDISPATCH pdispSelectionObjects = Exam.GetSelectionObjects();

if (pdispSelectionObjects)
{
   ISelectionObjects SelectionObjects;

   SelectionObjects.AttachDispatch( pdispSelectionObjects );

   for (long lSelectionObjects = 0; lSelectionObjects < SelectionObjects.GetCount();
               lSelectionObjects++)
   {
      LPDISPATCH pdispObject = SelectionObjects.GetItem( lSelectionObjects );

      if (pdispObject)
      {
         IObject Object;

         Object.AttachDispatch( pdispObject );
      }
   }
}
```

8.9 Image Object

The following table describes the object's methods:

Method	Return	Description
GetPalette	SCODE	Retrieves associated palette as SafeArray object
GetPixelMap	SCODE	Retrieves image pixel data
SetPixelMap	SCODE	Sets image pixel data
Image2Exam	void	Returns exam coordinates to given image coordinates

Exam2Image	void	Returns image coordinates to given exam coordinates

The following table describes the object's properties:

Property	Type	Access	Description
Width	long	r	Width in pixels
Height	long	r	Height in pixels
XpelsPerMeter	long	r	X resolution
YpelsPerMeter	long	r	Y resolution
BitsPerPixel	short	r	Pixel depth
Type	BSTR	r	Image type
Regio	BSTR	r	Image regio
Name	BSTR	r	Image name
TimeStamp	DATE	r	Image time stamp
Make	BSTR	r	Image creation modality or company
HW	BSTR	r	Hardware description
SW	BSTR	r	Software description
Site	BSTR	r	Site name
AutoCorr	BSTR	r	Image Correction parameters/description
XrayParam	BSTR	r	Image XRay parameters
Inverted	boolean	rw	Inversion flag
Brightness	long	rw	Brightness value
Contrast	long	rw	Brightness value
Calibrated	boolean	r	Calibration flag
CalibrationFactor	double	r	Calibration factor
LUT	object	r	Associated LUT object
FlippedHorz	boolean	r	Flipping flag
FlippedVert	boolean	r	Flipping flag
Diagnoses	object	r	Associated Diagmoses object
GreyScale	boolean	r	Grayscale flag
ROI	object	r	Associated ROI object
Angles	object	r	Associated Angles object
Lengths	object	r	Associated Lengths object
Cuts	object	r	Associated Cuts object
Objects	object	r	Associated Objects object
Rotation	double	r	Rotation degrees

8.9.1 Working with the Image object

The image object holds the information about the image being shown. Using this object you can retrieve the image parameters, edit the pixels or retrieve the overlay objects which have been defined for it like the diagnoses.

```
LPDISPATCH pdispImage = TakeFrame.GetImage();

if (pdispImage)
{
   // Image
   IImage Image;

   Image.AttachDispatch( pdispImage, FALSE );

   // Diagnoses
   LPDISPATCH pdispImageDiagnoses = Image.GetDiagnoses();

   if (pdispImageDiagnoses)
   {
      IDiagnoses Diagnoses;

      Diagnoses.AttachDispatch( pdispImageDiagnoses );

      for (long lImageDiagnoses = 0; lImageDiagnoses < Diagnoses.GetCount();
               lImageDiagnoses++)
      {
         // Diagnosis
         LPDISPATCH pdispImageDiagnosis = Diagnoses.GetItem( lImageDiagnoses );

         if (pdispImageDiagnosis)
         {
            IDiagnosis Diagnosis;

            Diagnosis.AttachDispatch( pdispImageDiagnosis, FALSE );
         }
      }
   }
}
```

The GetPixelMap() and SetPixelMap() methods allow image data editing. Assuming there is a class named CImage with members BYTE* m_pPixels and BITMAPINFO m_bmi:

```
// Copies the pixel data from the SIDEXIS Image object to an array of bytes.
BOOL CImage::FromSidexis( LPDISPATCH pdispImage )
{
   if (!pdispImage)
      return FALSE;

   IImage         Image;
   Image.AttachDispatch( pdispImage, FALSE );

   //---------------------------------------------------------------------------
   // Clear internal buffers
   //---------------------------------------------------------------------------
   Clear();

   //---------------------------------------------------------------------------
   // Save the bitmap information internally
   //---------------------------------------------------------------------------
   m_pbmi = (BITMAPINFO*) new BYTE[ sizeof(BITMAPINFO) + 255 * sizeof( RGBQUAD ) ];   //
               maximum size

   m_pbmi->bmiHeader.biSize          = sizeof( BITMAPINFOHEADER );
   m_pbmi->bmiHeader.biWidth         = Image.GetWidth();
   m_pbmi->bmiHeader.biHeight        = Image.GetHeight();
   m_pbmi->bmiHeader.biPlanes        = 1;
   m_pbmi->bmiHeader.biBitCount      = Image.GetBitsPerPixel();
   m_pbmi->bmiHeader.biCompression   = BI_RGB;   // not compressed
   m_pbmi->bmiHeader.biSizeImage     = m_pbmi->bmiHeader.biWidth * m_pbmi-
               >bmiHeader.biHeight * (m_pbmi->bmiHeader.biBitCount >> 3);
   m_pbmi->bmiHeader.biXPelsPerMeter = Image.GetXPelsPerMeter();
```

```
m_pbmi->bmiHeader.biYPelsPerMeter = Image.GetYPelsPerMeter();
m_pbmi->bmiHeader.biClrUsed       = 0;
m_pbmi->bmiHeader.biClrImportant  = 0;

//--------------------------------------------------------------------------
// Palette data
//--------------------------------------------------------------------------
if (m_pbmi->bmiHeader.biBitCount == 8)
{
   COleSafeArray sa;                    // Zum Auslesen des SafeArray
   long          lElements = 0;    // Zum Auslesen des SafeArray: Anzahl Bildpunkte
   long          lLBound   = 0;    // Zum Auslesen des SafeArray: Upper Bound
   long          lUBound   = 0;    // Zum Auslesen des SafeArray: Lower Bound
   COleVariant varPalette;

   (void) Image.GetPalette( &varPalette );

   RGBQUAD* pPaletteSrc = NULL; // Zum Auslesen des SafeArray: Palette

   sa.Attach( varPalette );

   sa.GetLBound( 1, &lLBound );
   sa.GetUBound( 1, &lUBound );
   sa.AccessData( (void**) &pPaletteSrc );// Get a pointer to the elements of the array
            and increments the lock count on the array

   lElements = lUBound - lLBound + 1;     // Get no. of elements in array

   // Inhalt kopieren
   (void) memcpy( m_pbmi->bmiColors, pPaletteSrc, sizeof( BYTE ) * lElements );

   sa.UnaccessData();                     // Decrement lock count
}
//--------------------------------------------------------------------------
// ROI (has to be rotated for matching the actual rotation of image)
//--------------------------------------------------------------------------
LPDISPATCH pdispROI = Image.GetRoi();

if (pdispROI)
{
   IROI ROI;
   long lLeft = 0, lTop = 0, lWidth = 0, lHeight = 0;

   ROI.AttachDispatch( pdispROI );
   ROI.GetPosition( &lLeft, &lTop, &lWidth, &lHeight );

   m_rcROI.left   = lLeft;
   m_rcROI.top    = lTop;
   m_rcROI.right  = lLeft + lWidth - 1;
   m_rcROI.bottom = lTop + lHeight - 1;

   ROI.ReleaseDispatch();

   // The ROI coordinates have to be rotated to match the image rotation
   int   nRotation = (int) Image.GetRotation();
   CRect rcRotated; // Rückgabe

   switch (nRotation)
   {
      case  90:

         rcRotated.left   =               m_rcROI.top;
         rcRotated.top    = GetHeight() - m_rcROI.right;
         rcRotated.right  =               m_rcROI.bottom;
         rcRotated.bottom = GetHeight() - m_rcROI.left;

         break;

      case 180:

         rcRotated.left   = GetWidth()  - m_rcROI.right;
         rcRotated.top    = GetHeight() - m_rcROI.bottom;
         rcRotated.right  = GetWidth()  - m_rcROI.left;
         rcRotated.bottom = GetHeight() - m_rcROI.top;
```

```
            break;

         case 270:

            rcRotated.left   = GetWidth() - m_rcROI.bottom;
            rcRotated.top    =              m_rcROI.left;
            rcRotated.right  = GetWidth() - m_rcROI.top;
            rcRotated.bottom =              m_rcROI.right;

            break;

         default:
            rcRotated = m_rcROI;
            break; // 0°
      }
      m_rcROI = rcRotated;
   }
   else
   {
      m_rcROI.left   = 0;
      m_rcROI.top    = 0;
      m_rcROI.right  = GetWidth();
      m_rcROI.bottom = GetHeight();
   }
   //---------------------------------------------------------------------------
   // Generate the device-independent bitmap
   //---------------------------------------------------------------------------
   COleVariant    varPixels;
   COleSafeArray  saPixels;            // Passed in Pixel Array
   BYTE*          pPixels    = NULL;   // Pixel Array for copying
   BYTE*          pDIBPixels = NULL;   // The pixel data pointer returned from
             CreateDIBSection
   CDC            dc;                  // Needed as CreateDIBSection parameter

   varPixels.Clear();

   dc.CreateCompatibleDC( NULL );

   (void) Image.GetPixelMap( &varPixels );   // Get pixel data from SIDEXIS

   saPixels.Attach( varPixels );             // Access the SafeArray Bytes
   saPixels.AccessData((LPVOID*)&pPixels);

   // Creation of the HBITMAP to hold the image data
   m_hBitmap = CreateDIBSection( dc.GetSafeHdc(), m_pbmi, DIB_RGB_COLORS, (void**)
               &pDIBPixels, NULL, 0 );
   memcpy( pDIBPixels, pPixels, m_pbmi->bmiHeader.biSizeImage );

   saPixels.UnaccessData();   //decrement lock count

   if (!m_hBitmap)
      return FALSE;

   GdiFlush();

   //---------------------------------------------------------------------------
   // Detach so the caller can work with the dispatch further
   //---------------------------------------------------------------------------
   (void) Image.DetachDispatch();
   return TRUE;
}
```

8.9.1.1 Pixel maps and palettes

The usage of pixel maps inside the image objects is pretty much the same as the Device-Independend Bitmap (DIB) handling of Windows.

The Get/SetPixelMap methods of the Image object only deal with the array of pixels.

For 256 color images (greyscale images) each byte corresponds to one pixel. The color of the pixel is obtained by using the 8-bit value as an index to the 256 entries of the associated palette.

Please keep in mind that SIDEXIS handles pixel rows as well as columns only as a multiple of 4 bytes.

In the current version only 8bit grayscale and 24bit true color images are supported. But note that future versions of SIDEXIS will work with grayscale images with more then 8bits per pixel!

The GetPalette method retrieves a byte array corresponding to the bmiColors member of a windows DIB.

For an example on how to work with palettes see sample code in chapter “Working with the Image object”.

8.9.2 Methods

8.9.2.1 GetPalette

Retrieves associated palette as SafeArray object

Definition	SCODE GetPalette(VARIANT* pvarPalette)
Return	COM errorcode
Params	pvarPalette : Palette safearray
C++ Usage	SCODE GetPalette(VARIANT* pvarPalette);
Comments	Please refer to MSDN and MFC documentations to get more information about COM programming with SafeArrays The palette is onyl returned in case of 8 bit grey scale images. All other types (BitsPerPixel != 8) will return an error.

8.9.2.2 GetPixelMap

Retrieves image pixel data

Definition	SCODE GetPixelMap(VARIANT* pvarPixelMap)
Return	COM errorcode
Params	PvarPixelMap : Pixelarray as safearray
C++ Usage	SCODE GetPixelMap(VARIANT* pvarPixelMap);
Comments	The pixels come raw, i.e. without any current brightness / contrast and other settings processed into! By contrast, when an image was rotated, the pixel map comes with rotation “already inside”! Please refer to MSDN and MFC documentations to get more information about COM programming with SafeArrays.

8.9.2.3 SetPixelMap

Sets image pixel data

Definition	SCODE SetPixelMap(VARIANT* pvarPixelMap)
Return	COM errorcode
Params	pvarPixelMap : Pixelarray as safearray

C++ Usage	SCODE SetPixelMap(VARIANT* pvarPixelMap);
Comments	Please refer to MSDN and MFC documentations to get more information about COM programming with SafeArrays

8.9.2.4 Image2Exam

Returns exam coordinates to given image coordinates

Definition	void Image2Exam(long lXImage, long lYImage, long* plXExam, long* plYExam)
Return	void
Params	lXImage : Image X ordinate lYImage : Image Y ordinate plXExam : resulting exam X ordinate plYExam : resulting exam Y ordinate
C++ Usage	void Image2Exam(long lXImage, long lYImage, long* plXExam, long* plYExam);
Comments	-

8.9.2.5 Exam2Image

Returns image coordinates to given exam coordinates

Definition	void Exam2Image(long lXExam, long lYExam, long* plXImage, long* plYImage)
Return	void
Params	lXExam : exam X ordinate lYExam : exam Y ordinate lpXImage : resulting Image X ordinate lpYImage : resulting Image Y ordinate
C++ Usage	void Exam2Image(long lXExam, long lYExam, long* plXImage, long* plYImage);
Comments	-

8.9.3 Properties

8.9.3.1 Width

Type	long
Access	read
C++ Usage	long GetWidth();
Comments	Remember that in the rotated case, “width” may apply to the vertical side of the pixel map!

8.9.3.2 Height

Type	Long
Access	Read

C++ Usage	Long GetHeight();
Comments	Remember that in the rotated case, “height” may apply to the horizontal side of the pixel map!

8.9.3.3 XPelsPerMeter

Type	long
Access	read
C++ Usage	long GetXPelsPerMeter();
Comments	-

8.9.3.4 YPelsPerMeter

Type	long
Access	read
C++ Usage	long GetYPelsPerMeter();
Comments	-

8.9.3.5 BitsPerPixel

Type	Short
Access	Read
C++ Usage	short GetBitsPerPixel();
Comments	Currently only 8bit (Greyscale images and 24bit (True colour images) possible!

8.9.3.6 Type

The corresponding image type

Type	BSTR
Access	Read
C++ Usage	CString GetType();
Comments	Examples: "XI" , "XP" Refer to 10.1 for a detailed description of SIDEXIS type definitions.

8.9.3.7 Regio

The corresponding image regio

Type	BSTR
Access	read
C++ Usage	CString GetRegio();
Comments	Refer to 10.1 for a detailed description of SIDEXIS type definitions.

8.9.3.8 Name

Type	BSTR
Access	read
C++ Usage	CString GetName();
Comments	-

8.9.3.9 TimeStamp

Type	DATE
Access	read
C++ Usage	DATE GetTimeStamp();
Comments	This is the time of image creation.

8.9.3.10 Make

Type	BSTR
Access	read
C++ Usage	CString GetMake();
Comments	-

8.9.3.11 HW

Type	BSTR
Access	read
C++ Usage	CString GetHw();
Comments	Reserved. Do not use this property in the current version.

8.9.3.12 SW

Type	BSTR
Access	Read
C++ Usage	CString GetSw();
Comments	Reserved. Do not use this property in the current version.

8.9.3.13 Site

Type	BSTR
Access	read
C++ Usage	CString GetSite();
Comments	-

8.9.3.14 AutoCorr

Type	BSTR
Access	read
C++ Usage	CString GetAutoCorr();
Comments	Reserved. Do not use this property in the current version.

8.9.3.15 XRayParam

Type	BSTR
Access	Read
C++ Usage	Cstring GetXRayParam();
Comments	Reserved. Do not use this property in the current version.

8.9.3.16 Inverted

Type	Boolean
Access	Read, write
C++ Usage	BOOL GetInverted(); void SetInverted(BOOL);
Comments	Indicates an inverted LUT image display in case of greyscale images.

8.9.3.17 Brightness

Type	Long
Access	Read
C++ Usage	long GetBrightness();
Comments	This attribute is a LUT/palette manipulating value. The bitmap pixel values are not modified. Range: [-255...0...255]

8.9.3.18 Contrast

Type	Long
Access	Read, write
C++ Usage	long GetContrast(); void SetBrightness(long);
Comments	This attribute is a LUT/palette manipulating value. The bitmap pixel values are not modified. Range: [-127...0...127]

8.9.3.19 Calibrated

Type	Boolean
Access	Read
C++ Usage	BOOL GetCalibrated();
Comments	Indicates an applied callibration factor. In this case the ->CallibrationFactor may be accessed to calculate real distances between points.

8.9.3.20 CalibrationFactor

Type	Double
Access	Read
C++ Usage	double GetCalibrationFactor();
Comments	Using the calibration length measurement tool inside SIDEXIS a calibration factor can be defined. This represents the the factor to get a real distance in mm for a given pixel distance.

8.9.3.21 LUT

Type	Object
Access	Read
C++ Usage	LPDISPATCH GetLut();
Comments	See 8.10 for LUT details

8.9.3.22 FlippedHorz

Type	Boolean
Access	Read
C++ Usage	BOOL GetFlippedHorz();
Comments	-

8.9.3.23 FlippedVert

Type	Boolean
Access	read
C++ Usage	BOOL GetFlippedVert();
Comments	-

8.9.3.24 Diagnoses

Type	object
Access	read
C++ Usage	LPDISPATCH GetDiagnoses();
Comments	See for 8.16 Diagnoses details

8.9.3.25 GreyScale

Type	boolean
Access	read
C++ Usage	BOOL GetGreyScale();
Comments	-

8.9.3.26 ROI

Type	object
Access	read
C++ Usage	LPDISPATCH GetRoi();
Comments	See 8.18 for ROI details

8.9.3.27 Angles

Type	object
Access	read
C++ Usage	LPDISPATCH GetAngles();
Comments	See 8.21 for Angles collection details.

8.9.3.28 Lengths

Type	object
Access	read
C++ Usage	LPDISPATCH GetLengths();
Comments	See 8.11 for Lengths collection details

8.9.3.29 Cuts

Type	object
Access	read
C++ Usage	LPDISPATCH GetCuts();
Comments	See 8.19 for Cuts collection details

8.9.3.30 Objects

Type	object
Access	read
C++ Usage	LPDISPATCH GetObjects();
Comments	See 8.6 for Objects collection details

8.9.3.31 Rotation

Type	double
Access	read
C++ Usage	double GetRotation();
Comments	-

8.10 LUT object

The LUT object holds information about a palette of an associated image.

The following table describes the object's properties:

Property	Type	Access	Description
Size	long	r	Holds number of palette entries
StockPalette	long	r	Stock palette ID

8.10.1 Working with the LUT object

```
LPDISPATCH pdispLUT = Image.GetLut();

if (pdispLUT)
{
   ILUT LUT;
   long lVal;

   LUT.AttachDispatch( pdispLUT );

   lVal = LUT.GetStockPalette();
}
```

8.10.2 Properties

8.10.2.1 Size

Type	long
Access	read
C++ Usage	long GetSize();
Comments	In the current version 8bit greyscale palette entries are supported only, the size value therefore is 256.

8.10.2.2 StockPalette

Stock palette ID

Type	long
Access	read
C++ Usage	long GetStockPalette();
Comments	In the current version a constant value of 2 will be returned. Reserved for future extensions.

8.11 Lengths collection

The Lengths collection allows the retrieval of existing Length objects on an Image object.

The Lengths collection operation is similar to the Objects collection operations, please refer 8.6 for details.

8.11.1 Working with the Lengths collection

```
LPDISPATCH pdispLengths = Image.GetLengths();

if (pdispLengths)
{
   ILengths Lengths;

   Lengths.AttachDispatch( pdispLengths );

   for (long lCountLengths = 0; lCountLengths < Lengths.GetCount(); lCountLengths++)
   {
      LPDISPATCH pdispLength = Lengths.GetItem( lCountLengths );

      if (pdispLength)
      {
         // Length
         ILength Length;

         Length.AttachDispatch( pdispLength, FALSE );
      }
   }
}
```

8.12 Length object

The following table describes the object's properties:

Property	Type	Access	Description
Points	object	r	Reference to an associated Points collection
Measure	double	r	Length value
MeasureUnit	long	r	Length measure unit [NGPixel, NGMicrometer,...]

8.12.1 Working with the Length object

The typical usage for the Length object is retrieving it's length.

```
LPDISPATCH pdispLength = Lengths.GetItem( lCountLengths );

if (pdispLength)
{
   // Length
   ILength Length;

   Length.AttachDispatch( pdispLength, FALSE );

   CString strMeasureUnit;

   switch (Length.GetMeasureUnit())
   {
      case NGPixel:
         strMeasureUnit = _T("Pixel");
         break;

      case NGMicrometer:
         strMeasureUnit = _T("Micrometer");
```

```
         break;

      case NGDegree:
         strMeasureUnit = _T("Degree");
         break;

      case NGPercent:
         strMeasureUnit = _T("Percent");
         break;

      default: break;
   }
   CString strMeasure;

   strMeasure.Format( _T("%f %s"), Length.GetMeasure(), strMeasureUnit );
}
```

8.12.2 Properties

8.12.2.1 Points

Type	object
Access	read
C++ Usage	LPDISPATCH GetPoints();
Comments	See 8.13 for Points collection details

8.12.2.2 Measure

Length value.

Type	double
Access	read
C++ Usage	double GetMeasure();
Comments	-

8.12.2.3 MeasureUnit

Length measure unit.

Type	long
Access	read
C++ Usage	long GetMeasureUnit();
Comments	The possible unit definitions are: NGPixel = 1, NGMicrometer = 2, NGDegree = 3, NGPercent = 4

8.13 Points collection

The points collection exists for various object types like Length, Arrow, Diagnosis, Cut, Angle, Polyline and Length. It can be used to retrieve the coordinates of the points which define the object.

The Points collection operations are similar to the Objects collection operations, please refer 8.6 for details.

Points coordinates can be used in any applied coordinate system. Please refer to the hosting object coordinate system environment for details.

8.13.1 Working with the Points collection

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

if (pdispPoints)
{
   IPoints Points;

   Points.AttachDispatch( pdispPoints );

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

      if (pdispPoint)
      {
         IPoint Point;

         Point.AttachDispatch( pdispPoint );

         long lX, lY;

         Point.Get( &lX, &lY );

         CString strPoint;

         strPoint.Format( _T("%d, %d"), lX, lY );
      }
   }
}
```

8.14 Point object

The following table describes the object's methods:

Method	Return	Description
Get	void	Returns Pixel coordinate

The following table describes the object's properties:

Property	Type	Access	Description
X	long	r	X ordinate
Y	long	r	Y ordinate

8.14.1 Working with the Point object

A single point object only returns it's coordinate. It can't be changed.

```
LPDISPATCH pdispPoint = Points.GetItem( lPoints );
```

```
if (pdispPoint)
{
   IPoint Point;

   Point.AttachDispatch( pdispPoint );

   long lX, lY;

   Point.Get( &lX, &lY );

   CString strPoint;

   strPoint.Format( _T("%d, %d"), lX, lY );
}
```

8.14.2 Methods

8.14.2.1 Get

Returns Pixel coordinate

Definition	void Get(long* plXCoordinate, long* plYCoordinate)
Return	void
Params	plXCoordinate : X ordinate plYCoordinate : Y ordinate
C++ Usage	void Get(long* plXCoordinate, long* plYCoordinate);
Comments	-

8.14.3 Properties

8.14.3.1 X

Type	long
Access	read
C++ Usage	long GetX();
Comments	-

8.14.3.2 Y

Type	long
Access	read
C++ Usage	long GetY();
Comments	-

8.15 Arrow object

The arrow object can connect objects like Images and Diagnoses. Therefore it can return it's connected objects at the beginning and end of it. The Arrow objects can be accessed through the Objects collections of Exam and Image objects.

Arrow objects with start- and endpoints inside the same image belong to that image. All other arrow objects belong to the Exam->Objects collection.

The following table describes the object's methods:

Method	Return	Description
GetObjectAttachedToBegin	object	Retrieves object at arrow's beginning
GetObjectAttachedToEnd	object	Retrieves object at arrow's end

The following table describes the object's properties:

Property	Type	Access	Description
Points	object	r	Reference to the Points collection of all arrow describing points

8.15.1 Working with the Arrow object

```
LPDISPATCH pdispObjects = Exam.GetObjects();

if (pdispObjects)
{
   IObjects Objects;

   Objects.AttachDispatch( pdispObjects );

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

      if (pdispObject)
      {
         IObject Object;

         Object.AttachDispatch( pdispObject );

         switch (Object.GetType())
         {
            case NGArrow:
            {
               LPDISPATCH pdispArrow = Object.GetTypedObject();

               if (pdispArrow)
               {
                  IArrow Arrow;

                  Arrow.AttachDispatch( pdispArrow );

                  LPUNKNOWN pUnk = NULL;

                  long lX, lY;

                  LPDISPATCH pdispObjectAtBegin = Arrow.GetObjectAttachedToBegin( &lX, &lY
               );
               if (pdispObjectAtBegin)
               {
                   if (SUCCEEDED( pdispObjectAtBegin->QueryInterface( IID_IImage, (void**)
               &pUnk ) ))
                   {
                       IImage ImageArrow;

                       ImageArrow.AttachDispatch( pdispObjectAtBegin );
                       pUnk->Release();
                   }
                   else if (SUCCEEDED( pdispObjectAtBegin->QueryInterface( IID_IDiagnosis,
               (void**) &pUnk ) ))
                   {
                      IDiagnosis DiagnosisArrow;

                      DiagnosisArrow.AttachDispatch( pdispObjectAtBegin );
```

```
                pUnk->Release();
            }
          }
          LPDISPATCH pdispObjectAtEnd = Arrow.GetObjectAttachedToEnd( &lX, &lY );

          if (pdispObjectAtEnd)
          {
             if (SUCCEEDED( pdispObjectAtEnd->QueryInterface( IID_IImage, (void**)
          &pUnk ) ))
             {
                 IImage ImageArrow;

                 ImageArrow.AttachDispatch( pdispObjectAtEnd );
                 pUnk->Release();
             }
             else if (SUCCEEDED( pdispObjectAtEnd->QueryInterface( IID_IDiagnosis,
          (void**) &pUnk ) ))
             {
                IDiagnosis DiagnosisArrow;

                DiagnosisArrow.AttachDispatch( pdispObjectAtEnd );
                pUnk->Release();
             }
          }
          break;
       }
       default: break;
    }
  }
 }
}
```

8.15.2 Methods

8.15.2.1 GetObjectAttachedToBegin

Definition	IDispatch* GetObjectAttachedToBegin(long* plX, long* plY)
Return	Beginning object reference
Params	plX : X ordinate plY : Y ordinate
C++ Usage	LPDISPATCH GetObjectAttachedToBegin(long* plX, long* plY);
Comments	The referenced coordinate system depends on the object context used.

8.15.2.2 GetObjectAttachedToEnd

Definition	IDispatch* GetObjectAttachedToEnd(long* plX, long* plY)
Return	End object reference
Params	plX : X ordinate [exam coordinate system] plY : Y ordinate [exam coordinate system]
C++ Usage	LPDISPATCH GetObjectAttachedToEnd(long* plX, long* plY);
Comments	The referenced coordinate system depends on the object context used.

8.15.3 Properties

8.15.3.1 Points

Holds a reference to the Points collection of all arrow describing points.

Type	object
Access	read
C++ Usage	LPDISPATCH GetPoints();
Comments	See 8.13 for Points collection details.

8.16 Diagnoses collection

The Diagnoses collection operations are similar to the Objects collection operations, please refer 8.6 for details.

8.16.1 Working with the Diagnoses collection

```
LPDISPATCH pdispImageDiagnoses = Image.GetDiagnoses();

if (pdispImageDiagnoses)
{
   IDiagnoses Diagnoses;

   Diagnoses.AttachDispatch( pdispImageDiagnoses );

   for (long lImageDiagnoses = 0; lImageDiagnoses < Diagnoses.GetCount();
               lImageDiagnoses++)
   {
      // Diagnosis
      LPDISPATCH pdispImageDiagnosis = Diagnoses.GetItem( lImageDiagnoses );

      if (pdispImageDiagnosis)
      {
         IDiagnosis Diagnosis;

         Diagnosis.AttachDispatch( pdispImageDiagnosis, FALSE );
      }
   }
}
```

8.17 Diagnosis object

The Diagnosis object represents the diagnosis for the image.

The following table describes the object's methods:

Method	Return	Description
GetObjectAttachedTo	object	Returns object connected to this diagnosis
GetPosition	void	Returns Diagnosis' position

The following table describes the object's properties:

Property	Type	Access	Description
Points	object	R	Diagnosis refers to these points
Comment	BSTR	R	Comment text

Regio	BSTR	R	Regio the Diagnosis deals with

8.17.1 Working with the Diagnosis object

```
LPDISPATCH pdispImageDiagnosis = Diagnoses.GetItem( lImageDiagnoses );

if (pdispImageDiagnosis)
{
   IDiagnosis Diagnosis;

   Diagnosis.AttachDispatch( pdispImageDiagnosis, FALSE );

   CString strComment = Diagnosis.GetComment();
}
```

8.17.2 Methods

8.17.2.1 GetObjectAttachedTo

Definition	IDispatch* GetObjectAttachedTo(long* plX, long* plY)
Return	Attached object reference
Params	plX : X ordinate [exam coordinate system] plY : Y ordinate [exam coordinate system]
C++ Usage	LPDISPATCH GetObjectAttachedTo(long* plX, long* plY);

8.17.2.2 GetPosition

Retrieves Diagnosis position and size.

Definition	void GetPosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);
Return	Void
Params	pnLeft : top left ordinate [exam coordinate system] pnTop : top left ordinate [exam coordinate system] pnWidth: width in pixels pnHeight: height in pixels
C++ Usage	void GetPosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);
Comments	A width of 0 indicates an iconized display of the diagnosis.

8.17.3 Properties

8.17.3.1 Points

Type	object
Access	read
C++ Usage	LPDISPATCH GetPoints();
Comments	Refer to 8.13 for Points collection details.

8.17.3.2 Comment

Type	BSTR
Access	read
C++ Usage	CString GetComment();
Comments	-

8.17.3.3 Regio

Type	BSTR
Access	read
C++ Usage	CString GetRegio();
Comments	Regio of tooth, see 10.1 for details.

8.18 ROI object

The ROI object returns the coordinates of the ROI if one has been set.

The following table describes the object's methods:

Method	Return	Description
GetPosition	void	Returns ROI position and size

8.18.1 Working with the ROI object

```
LPDISPATCH pdispROI = Image.GetRoi();

if (pdispROI)
{
   IROI ROI;

   ROI.AttachDispatch( pdispROI );

   long lLeft, lTop, lWidth, lHeight;

   ROI.GetPosition( &lLeft, &lTop, &lWidth, &lHeight );
}
```

8.18.2 Methods

8.18.2.1 GetPosition

Returns ROI position and size

Definition	void GetPosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);
Return	void
Params	pnLeft : top left ordinate pnTop : top left ordinate pnWidth: width in pixels pnHeight: height in pixels

C++ Usage	void GetPosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);

8.19 Cuts collection

The Cuts collection operations are similar to the Objects collection operations, please refer 8.6 for details.

8.20 Cut object

The following table describes the object's properties:

Property	Type	Access	Description
Points	object	r	Refers to associated Points collection

8.20.1 Properties

8.20.1.1 Points

Refers to associated Points collection

Type	object
Access	read
C++ Usage	LPDISPATCH GetPoints();
Comments	Refer 8.13 for a detailed Points collection description.

8.21 Angles collection

The Angles collection operations are similar to the Objects collection operations, please refer 8.6 for details.

8.21.1 Working with the Angles collection

```
LPDISPATCH pdispImageAngles = Image.GetAngles();

if (pdispImageAngles)
{
   IAngles Angles;

   Angles.AttachDispatch( pdispImageAngles );

   for (long lImageAngles = 0; lImageAngles < Angles.GetCount(); lImageAngles++)
   {
      LPDISPATCH pdispImageAngle = Angles.GetItem( lImageAngles );

      if (pdispImageAngle)
      {
         IAngle Angle;

         Angle.AttachDispatch( pdispImageAngle, FALSE );
      }
   }
}
```

8.22 Angle object

The following table describes the object's properties:

Property	Type	Access	Description
Points	object	r	Refers to associated Points collection

8.22.1 Properties

8.22.1.1 Points

Refers to associated Points collection

Type	object
Access	read
C++ Usage	LPDISPATCH GetPoints();
Comments	Refer 8.13 for a detailed Points collection description.

8.23 Polyline object

The following table describes the object's properties:

Property	Type	Access	Description
Points	object	r	Refers to associated Points collection

8.23.1 Properties

8.23.1.1 Points

Refers to associated Points collection

Type	object
Access	read
C++ Usage	LPDISPATCH GetPoints();
Comments	Refer 8.13 for a detailed Points collection description.

8.24 Symbol object

The following table describes the object's methods:

Method	Return	Description
GetPosition	void	Returns Symbol position and size

The following table describes the object's properties:

Property	Type	Access	Description
Type	BSTR	r	Type string
Label	BSTR	r	Label string

8.24.1 Working with the Symbol object

The Symbol objects is accessible through the Objects collection of an Image object.

```
LPDISPATCH pdispObjects = Image.GetObjects();

if (pdispObjects)
{
   IObjects Objects;

   Objects.AttachDispatch( pdispObjects );

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

      if (pdispObject)
      {
         IObject Object;

         Object.AttachDispatch( pdispObject );

         switch (Object.GetType())
         {
            case NGSymbol:
            {
              LPDISPATCH pdispSymbol = Object.GetTypedObject();

               if (pdispSymbol)
               {
                  ISymbol Symbol;

                  Symbol.AttachDispatch( pdispSymbol );

                  long lLeft, lTop, lWidth, lHeight;

                  Symbol.GetPosition( &lLeft, &lTop, &lWidth, &lHeight );
               }
               break;
            }
            default: break;
         }
      }
   }
}
```

8.24.2 Methods

8.24.2.1 GetPosition

Returns Symbol position and size

Definition	void GetPosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);
Return	void
Params	pnLeft : top left ordinate [exam coordinate system] pnTop : top left ordinate [exam coordinate system] pnWidth: width in pixels pnHeight: height in pixels
C++ Usage	void GetPosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);

8.24.3 Properties

8.24.3.1 Type

Type	BSTR
Access	read
C++ Usage	CString GetType();
Comments	The only symbol supported in the current release is the cross symbol ("NGCross")

8.24.3.2 Label

Type	BSTR
Access	read
C++ Usage	CString GetLabel();
Comments	-

8.25 Rect object

The following table describes the object's methods:

Method	Return	Description
GetPosition	void	Returns Rect position and size

8.25.1 Working with the Rect object

The Rect objects are accessible through the Objects collection of an Image object.

```
LPDISPATCH pdispObjects = Image.GetObjects();

if (pdispObjects)
{
   IObjects Objects;

   Objects.AttachDispatch( pdispObjects );

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

      if (pdispObject)
      {
         IObject Object;

         Object.AttachDispatch( pdispObject );

         switch (Object.GetType())
         {
            case NGRect:
            {
              LPDISPATCH pdispRect = Object.GetTypedObject();

               if (pdispRect)
               {
                  IRect Rect;
```

```
                Rect.AttachDispatch( pdispRect );

                long lLeft, lTop, lWidth, lHeight;

                Rect.GetPosition( &lLeft, &lTop, &lWidth, &lHeight );
             }
             break;
          }
          default: break;
       }
    }
 }
}
```

8.25.2 Methods

8.25.2.1 GetPosition

Returns Symbol position and size

Definition	void GetPosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);
Return	Void
Params	pnLeft : top left ordinate [exam coordinate system] pnTop : top left ordinate [exam coordinate system] pnWidth: width in pixels pnHeight: height in pixels
C++ Usage	void GetPosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);

8.26 Ellipse object

The Ellipse objects are accessible through the Objects collection of an Image object.

The following table describes the object's methods:

Method	Return	Description
GetPosition	Void	Returns Ellipse position and framing rectangle size

8.26.1 Working with the Elipse object

```
LPDISPATCH pdispObjects = Image.GetObjects();

if (pdispObjects)
{
   IObjects Objects;

   Objects.AttachDispatch( pdispObjects );

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

      if (pdispObject)
      {
         IObject Object;

         Object.AttachDispatch( pdispObject );

         switch (Object.GetType())
         {
            case NGEllipse:
```

```
            {
              LPDISPATCH pdispEllipse = Object.GetTypedObject();

               if (pdispEllipse)
               {
                  IEllipse Ellipse;

                  Ellipse.AttachDispatch( pdispEllipse );

                  long lLeft, lTop, lWidth, lHeight;

                  Ellipse.GetPosition( &lLeft, &lTop, &lWidth, &lHeight );
               }
               break;
            }
            default: break;
         }
      }
   }
}
```

8.26.2 Methods

8.26.2.1 GetPosition

Returns Ellipse position and size

Definition	void GetPosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);
Return	void
Params	pnLeft : top left ordinate [exam coordinate system] pnTop : top left ordinate [exam coordinate system] pnWidth: width in pixels pnHeight: height in pixels
C++ Usage	void GetPosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);

8.27 Text object

This object is used to manage text annotation.

The following table describes the object's methods:

Method	Return	Description
GetPosition	void	Returns Ellipse position and framing rectangle size

The following table describes the object's properties:

Property	Type	Access	Description
Comment	BSTR	r	The annotation string

8.27.1 Methods

8.27.1.1 GetPosition

Returns Ellipse position and size

Definition	void GetPosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);

Return	void
Params	pnLeft : top left ordinate [exam coordinate system] pnTop : top left ordinate [exam coordinate system] pnWidth: width in pixels pnHeight: height in pixels
C++ Usage	void GetPosition(long* pnLeft, long* pnTop, long* pnWidth, long* pnHeight);

8.27.2 Properties

8.27.2.1 Comment

Type	BSTR
Access	read
C++ Usage	CString GetComment();
Comments	-

9 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!

10 Appendix

10.1 Image types and regios

The following definitions apply mainly to the Type and Regio properties of Image and TakeFrame objects.

10.1.1 Type

The type of an image is coded in a string. Currently, the code consists of a pair of 2 characters. The first character represents the type of acquisition technique ('X' for X-ray, 'V' for visible light), whilst the second defines an acquisition subtype.

Type	Meaning
"??"	Any image, acquisition technique unknown
"X?"	X-ray of unknown
"XI"	Intraoral X-ray
"XP"	Panoramic X-ray
"XC"	Cephalometric X-ray
"XS"	Transversal slices X-ray
"V?"	Any visible light image, subtype unknown
"VI"	Intraoral visible light image
"VC"	Cerec camera image (2D)
"VF"	Frontal view visible light image

10.1.2 Regio

The anatomic region covered by an image is coded in a string. The meaning of this code depends on the contents of the *Type* string. The length of Regio may be extended in future versions!

Type	Regio	Meaning
"??" "X?" "V?" "VF" <Any other>	"~~"	Not diagnosable
	" " (2 blanks)	Other / Unknown
"XI" "XS" "VI" "VC"	<##>	2-digit number = International tooth number (FDI tooth scheme, DIN EN ISO 3950) of the imaged tooth or one of the imaged teeth. Supernumerary teeth should be referenced by the next "normal" tooth, e.g. "28" for tooth 29, with a corresponding entry in a reason / notes / comments / findings field.

Type	Regio	Meaning
“XP”	<##>	2-digit number = Program number of the Orthophos:
	"01"	P1: Normal OPG
	"02"	P2: Normal OPG, reduced
	"03"	S2: Paranasal sinuses
	"04"	TM3: Temporomandibular joints lateral, ascending branches
	"05"	TM4: Temporomandibular joints p.a., ascending branches
	"06"	TM1: Lateral view of the temporomandibular joints, closed / open mouth
	"07"	TM2: P.a. view of the temporomandibular joints, closed / open mouth
	"08"	TM5: Lateral multi-layer image of the temporomandibular joints
	"09"	TM6: P.a. multi-layer image of the temporomandibular joints
	"10"	P10: Normal OPG for children
	"11"	P1: Normal OPG with constant magnification
	"12"	P12: Thick layer, anterior region
	"13"	S1: Maxillary sinuses, 2 images
	"14"	P1: Normal OPG, left side
	"15"	P1: Normal OPG, right side
	"16"	TS1: Multi-layer image, lateral teeth
“XC”	"01"	Front view from behind (PA)
	"02"	Front view from the front (AP)
	"03"	Lateral view
	"04"	Teleradiographic hand image

10.1.3 Examples of Type / Regio combinations

Type	Regio	Meaning
"XI"	"41"	Intraoral X-ray image of tooth 41
"XI"	"53"	Intraoral X-ray image of tooth 53 (denture/milk teeth)
"XP"	"01"	Panoramic tomogram with Orthophos Program 1
"XC"	"03"	Teleradiographic image, lateral view

10.2 Object types

```
//----------------------------------------------------------------------------
/// Headername  : NGObjTypes.h @file
///
///               SIDEXIS NG Programming Model object types definitions
///
/// © 2003 Sirona Dental Systems GmbH - All Rights Reserved
///
/// Status      : Released
///
/// Change history - Version 1.1
///
/// 26.11.02 - Draft => V.1.0 (MCI, Fa. MGW)
/// 19.01.03 - Review + Release V.1.0 (JueZi, GBE)
/// 14.03.03 - Added Polyline, Rect, Ellipse, Symbol => V.1.1 (MCI, Fa. MGW)
/// 14.03.03 - Renamed NGMeasure => NGLength, NGTextbox => NGText (MCI, Fa. MGW)
/// 15.03.03 - Review + Release V.1.1 (JueZi, GBE)
//----------------------------------------------------------------------------
#ifndef __NGOBJTYPES_H
#define __NGOBJTYPES_H

//----------------------------------------------------------------------------
// Definitions, macros, constants
//----------------------------------------------------------------------------
#if _MSC_VER > 1000 // VC++ Version > 4.0
#pragma once // open only once during build
#endif // _MSC_VER > 1000

//----------------------------------------------------------------------------
// Data types
//----------------------------------------------------------------------------
typedef enum
{
   NGNone        = -1,
   NGUnknown     =  0,
   NGTake        =  1,
   NGDiagnosis   =  2,
   NGArrow       =  3,
   NGRoi         =  4,
   NGLength      =  5,
   NGText        =  6,
   NGAngle       =  7,
   NGCut         =  8,
   NGImage       =  9,
   NGExam        = 10,
   NGPolyline    = 11,
   NGRect        = 12,
   NGEllipse     = 13,
   NGSymbol      = 14
} NGObjectTypes;

typedef enum
{
   NGPixel      = 1,
   NGMicrometer = 2,
   NGDegree     = 3,
   NGPercent    = 4
}  NGMeasureUnits;

#endif //__NGOBJTYPES_H
```

10.3 Interface IIDs

```
//----------------------------------------------------------------------------
/// Headername  : NGIIDs.h @file
///
///               COM Interface ID definitions for programming model types
///
/// © 2003 Sirona Dental Systems GmbH - All Rights Reserved
///
/// Status      : Released
///
/// Änderungshistorie - Version 1.1
///
/// - 20.01.03 - Draft V.1.0 (MCI, Fa. MGW)
/// - 20.01.03 - Review + Release V.1.0 (JueZi)
/// - 14.03.03 - Added Polyline, Text, Rect, Ellipse, Symbol => V.1.1 (MCI, Fa. MGW)
/// - 15.03.03 - Review + Release V.1.1 (JueZi)
//----------------------------------------------------------------------------
#ifndef __NGIIDS_H
#define __NGIIDS_H

//----------------------------------------------------------------------------
// Definitions
//----------------------------------------------------------------------------
#if _MSC_VER > 1000 // VC++ Version > 4.0
#pragma once // open only once during build
#endif // _MSC_VER > 1000

static const IID IID_IApplication       = { 0xa5176795, 0xb8fc, 0x11d6, { 0x88, 0xba, 0x0,
              0x50, 0xbf, 0x6, 0xb6, 0x65 } };    // {A5176795-B8FC-11D6-88BA-0050BF06B665}
static const IID IID_ISidexisNGDoc      = { 0x676EABDF, 0x691E, 0x4dd4, { 0x90, 0x3d, 0xc7,
              0x63, 0x27, 0x18, 0xcb, 0x3e } };  // {676EABDF-691E-4DD4-903D-C7632718CB3E}
static const IID IID_IView              = { 0xC7CFF233, 0xD762, 0x49AE, { 0x97, 0x46, 0xc2,
              0x20, 0x0e, 0xDB, 0x9a, 0x83 } };  // {C7CFF233-D762-49AE-9746-C2200EDB9A83}
static const IID IID_IWindow            = { 0x1913C583, 0x7933, 0x498F, { 0xa9, 0xd9, 0x55,
              0x02, 0x8d, 0x13, 0x97, 0x11 } };  // {1913C583-7933-498F-A9D9-55028D139711}

static const IID IID_IExam              = { 0x6a6b3b92, 0x1073, 0x4948, { 0x82, 0xf5, 0xf8,
              0x79, 0xca, 0xc1, 0xdf, 0x46 } };  // {6A6B3B92-1073-4948-82F5-F879CAC1DF46}
static const IID IID_IObject            = { 0x3ae73303, 0x4aeb, 0x4183, { 0x9e, 0xf5, 0x70,
              0xb7, 0x25, 0x4c, 0xbf, 0x26 } };  // {3AE73303-4AEB-4183-9EF5-70B7254CBF26}
static const IID IID_IPatient           = { 0xa1abad1b, 0xcc3a, 0x46c1, { 0xbf, 0xf7, 0x3b,
              0x54, 0xdd, 0x4f, 0xa5, 0xfe } };  // {A1ABAD1B-CC3A-46C1-BFF7-3B54DD4FA5FE}
static const IID IID_IArrow             = { 0x9865afd8, 0x65f6, 0x4d0e, { 0x9e, 0x5, 0x45,
              0xe6, 0xfb, 0xd1, 0x40, 0x19 } };   // {9865AFD8-65F6-4D0E-9E05-45E6FBD14019}
static const IID IID_ITakeFrame         = { 0x5ddab1a6, 0xe1b5, 0x11d6, { 0xb6, 0x24, 0x0,
              0x1, 0x2, 0x21, 0x6, 0xc8 } };      // {5DDAB1A6-E1B5-11D6-B624-0001022106C8}
static const IID IID_IImage             = { 0x21864ecf, 0x9041, 0x4dc8, { 0xba, 0x85, 0x87,
              0x82, 0x44, 0xea, 0x9e, 0xdf } };  // {21864ECF-9041-4DC8-BA85-878244EA9EDF}
static const IID IID_IROI               = { 0xe19e29ff, 0x7296, 0x41a0, { 0xaa, 0x5b, 0x24,
              0xdf, 0x81, 0x43, 0x15, 0x1d } };  // {E19E29FF-7296-41A0-AA5B-24DF8143151D}
static const IID IID_IAngle             = { 0x39287d2e, 0xcd85, 0x464e, { 0x91, 0x7c, 0x42,
              0x6a, 0x40, 0x39, 0x1b, 0xe9 } };  // {39287D2E-CD85-464E-917C-426A40391BE9}
static const IID IID_ICut               = { 0x779417a5, 0x597a, 0x4957, { 0xb8, 0x11, 0x5e,
              0x47, 0x20, 0xe3, 0xa2, 0x6d } };  // {779417A5-597A-4957-B811-5E4720E3A26D}
static const IID IID_IDiagnosis         = { 0x945eb694, 0x7a58, 0x4f30, { 0xad, 0xde, 0x69,
              0x56, 0xa5, 0x5b, 0x5b, 0x52 } };  // {945EB694-7A58-4F30-ADDE-6956A55B5B52}
static const IID IID_ILength            = { 0x8f24786d, 0x95b, 0x44a1, { 0xb1, 0xb3, 0x3b,
              0x3e, 0xb5, 0x3f, 0x4a, 0x8a } };   // {8F24786D-095B-44A1-B1B3-3B3EB53F4A8A}
static const IID IID_ILUT               = { 0xee5dc938, 0x5b78, 0x4a26, { 0x97, 0xc0, 0xa9,
              0x36, 0x25, 0x73, 0x35, 0x67 } };  // {EE5DC938-5B78-4A26-97C0-A93625733567}
static const IID IID_IPoint             = { 0xa92dbe03, 0x9225, 0x4daf, { 0x89, 0x52, 0x49,
              0xf3, 0x40, 0x1f, 0xb, 0xa5 } };   // {A92DBE03-9225-4daf-8952-49F3401F0BA5}
static const IID IID_IProperty          = { 0x6c6268cc, 0x4c07, 0x4d35, { 0x96, 0x72, 0xd7,
              0xe5, 0xb7, 0xc7, 0x35, 0x38 } };  // {6C6268CC-4C07-4d35-9672-D7E5B7C73538}
static const IID IID_IPolyline          = { 0x3f63eb68, 0x8e7a, 0x4116, { 0x98, 0xd3, 0x54,
              0x7, 0xd0, 0x12, 0x41, 0xc8 } };   // {3F63EB68-8E7A-4116-98D3-5407D01241C8}
static const IID IID_IText              = { 0xc06183c3, 0x942, 0x47b7, { 0x82, 0x2, 0x64,
              0x22, 0x33, 0x30, 0x61, 0x8f } };    // {C06183C3-0942-47B7-8202-
              64223330618F}
static const IID IID_IRect              = { 0x954285a, 0x3900, 0x4120, { 0x9f, 0x2, 0x5a,
              0x63, 0xd0, 0xf4, 0xc8, 0x79 } };    // {0954285A-3900-4120-9F02-
              5A63D0F4C879}
```

```
static const IID IID_IEllipse           = { 0xdd3f1d3f, 0x21c8, 0x44a7, { 0x88, 0x72, 0x24,
              0x9a, 0x7a, 0x1, 0xa3, 0xd4 } };   // {DD3F1D3F-21C8-44A7-8872-249A7A01A3D4}
static const IID IID_ISymbol            = { 0x37a086be, 0xbbfc, 0x4fa7, { 0xba, 0x48, 0xd,
              0xa0, 0xb6, 0x86, 0x16, 0x38 } };   // {37A086BE-BBFC-4FA7-BA48-0DA0B6861638}

static const IID IID_IAngles            = { 0x64c4b246, 0x9232, 0x461d, { 0xb0, 0x81, 0x96,
              0x9f, 0x27, 0xeb, 0xf1, 0x55 } };  // {64C4B246-9232-461D-B081-969F27EBF155}
static const IID IID_ICuts              = { 0xe808e625, 0x817b, 0x4ef4, { 0xbb, 0x10, 0x9e,
              0x5b, 0x43, 0x82, 0x61, 0x74 } };  // {E808E625-817B-4EF4-BB10-9E5B43826174}
static const IID IID_IDiagnoses         = { 0x7b874105, 0x7139, 0x4e9e, { 0xb2, 0xa5, 0xfb,
              0x1b, 0xca, 0x2d, 0xbc, 0x5 } };   // {7B874105-7139-4E9E-B2A5-FB1BCA2DBC05}
static const IID IID_ILengths           = { 0x54784f13, 0x2468, 0x47ba, { 0x8a, 0xbc, 0x8,
              0x87, 0x80, 0xc0, 0x4f, 0xb0 } };   // {54784F13-2468-47BA-8ABC-088780C04FB0}
static const IID IID_IProperties        = { 0xe2c3da53, 0xf615, 0x4ba9, { 0xab, 0xc8, 0xe6,
              0x61, 0x4e, 0x15, 0xd3, 0x1c } };  // {E2C3DA53-F615-4ba9-ABC8-E6614E15D31C}
static const IID IID_ITakeFrames        = { 0xadef6e34, 0x5782, 0x4252, { 0x8a, 0x37, 0x17,
              0x85, 0x67, 0xa5, 0xcb, 0x2c } };  // {ADEF6E34-5782-4252-8A37-178567A5CB2C}
static const IID IID_IExams             = { 0x8d28b11, 0x9c23, 0x4207, { 0xb8, 0x5f, 0x46,
              0xf5, 0x13, 0xab, 0x6d, 0xe9 } };   // {08D28B11-9C23-4207-B85F-46F513AB6DE9}
static const IID IID_IWindows           = { 0xdd594861, 0x130e, 0x4222, { 0x81, 0x92, 0x66,
              0x66, 0x46, 0x8e, 0xd7, 0x81 } };  // {DD594861-130E-4222-8192-6666468ED781}
static const IID IID_IObjects           = { 0xd7425545, 0x80b9, 0x4418, { 0x9b, 0x66, 0xa7,
              0xeb, 0x81, 0x8, 0xa6, 0x67 } };   // {D7425545-80B9-4418-9B66-A7EB8108A667}
static const IID IID_ISelectionObjects  = { 0x3988775c, 0x8253, 0x4c25, { 0x9d, 0x5f, 0xd9,
              0x7b, 0x29, 0xe6, 0x6b, 0x36 } };  // {3988775C-8253-4C25-9D5F-D97B29E66B36}
static const IID IID_IPoints            = { 0xae83bbf4, 0x3b43, 0x4477, { 0x90, 0xca, 0x83,
              0x12, 0x93, 0x7, 0x3c, 0x37 } };   // {AE83BBF4-3B43-4477-90CA-831293073C37}

#endif //__NGIIDS_H
```

10.4 Language IDs

```
//---------------------------------------------------------------------------
/// Headername  : lcid.h @file
///
///               Definition of Locale IDs.
///
/// © 2003 Sirona Dental Systems GmbH - All Rights Reserved
///
/// Status      : Released
///
/// Revision history - Version 1.0
///
/// - 16.01.03 - Draft => V.1.0 (MCI, Fa. MGW)
/// - 18.01.03 - Review + Release V.1.0 (JueZi)
//---------------------------------------------------------------------------
#ifndef __LCID_H
#define __LCID_H

//---------------------------------------------------------------------------
// Standard includes
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Own includes
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Definitions, macros und constants
//---------------------------------------------------------------------------
#if _MSC_VER > 1000 // VC++ Version > 4.0
#pragma once // open only once during build
#endif // _MSC_VER > 1000

#define LANGUAGE_GERMAN                 MAKELCID( MAKELANGID( LANG_GERMAN,
               SUBLANG_GERMAN ), SORT_DEFAULT )
#define LANGUAGE_ENGLISH_US             MAKELCID( MAKELANGID( LANG_ENGLISH,
               SUBLANG_ENGLISH_US ), SORT_DEFAULT )
#define LANGUAGE_ENGLISH_GB             MAKELCID( MAKELANGID( LANG_ENGLISH,
               SUBLANG_ENGLISH_UK ), SORT_DEFAULT )
#define LANGUAGE_FRENCH                 MAKELCID( MAKELANGID( LANG_FRENCH,
               SUBLANG_FRENCH ), SORT_DEFAULT )
#define LANGUAGE_FRENCH_BELGIAN         MAKELCID( MAKELANGID( LANG_FRENCH,
               SUBLANG_FRENCH_BELGIAN ), SORT_DEFAULT )
#define LANGUAGE_SPANISH                MAKELCID( MAKELANGID( LANG_SPANISH,
               SUBLANG_SPANISH ), SORT_DEFAULT )
#define LANGUAGE_ITALIAN                MAKELCID( MAKELANGID( LANG_ITALIAN,
               SUBLANG_ITALIAN ), SORT_DEFAULT )
#define LANGUAGE_DUTCH                  MAKELCID( MAKELANGID( LANG_DUTCH,      SUBLANG_DUTCH
               ), SORT_DEFAULT )
#define LANGUAGE_DUTCH_BELGIAN          MAKELCID( MAKELANGID( LANG_DUTCH,
               SUBLANG_DUTCH_BELGIAN ), SORT_DEFAULT )
#define LANGUAGE_RUSSIAN                MAKELCID( MAKELANGID( LANG_RUSSIAN,
               SUBLANG_DEFAULT ), SORT_DEFAULT )
#define LANGUAGE_JAPANESE               MAKELCID( MAKELANGID( LANG_JAPANESE,
               SUBLANG_DEFAULT ), SORT_DEFAULT )
#define LANGUAGE_KOREAN                 MAKELCID( MAKELANGID( LANG_KOREAN,
               SUBLANG_KOREAN ), SORT_KOREAN_KSC )
#define LANGUAGE_ARABIC_SAUDIARABIA     MAKELCID( MAKELANGID( LANG_ARABIC,
               SUBLANG_ARABIC_SAUDI_ARABIA ), SORT_DEFAULT )

//---------------------------------------------------------------------------
// Data types
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Prototype of the public functions
//---------------------------------------------------------------------------

#endif //__LCID_H
```

10.5 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.

11 Index