



ENOVIA SmarTeam | Dassault Systèmes

www.smartteam.com

www.3ds.com

ENOVIA SmarTeam

SMARTTEAM OBJECT MODEL

PROGRAMMER'S GUIDE

Important Notice

© Dassault Systèmes, 2004, 2008. All rights reserved.

CATIA, ENOVIA, SmarTeam and the 3DS logo are registered trademarks of Dassault Systèmes or its subsidiaries in the US and/or other countries.

PROPRIETARY RIGHTS NOTICE: This documentation is the property of Dassault Systèmes. This documentation shall be treated as confidential information and may only be used by employees or contractors of the Customer in accordance with the terms of the End-User License Agreement accepted by Customer.

Any use of the Licensed Program contained in this media or accompanying it, is subject to the terms of the End User License Agreement accepted by Customer. The Licensed Program is protected by international copyright laws and international treaties. Unauthorized use, reproduction and/or distribution of any of the Licensed Program, or any part thereof, may result in severe civil and/or criminal penalties, and will be prosecuted to the maximum extent possible under the law. Company names and product names mentioned herein are the property of their respective owners and certain portions of the Licensed Program contain elements subject to copyright owned by these entities. See the Documentation CD provided with the Licensed Program for details and/or additional terms and conditions relating to these entities. Part No:

API-A3-1803007

Table of Contents

1.INTRODUCTION	1
What is the SmarTeam Object Model?	2
Basic Libraries and Service Libraries	3
Engine and Session Objects	4
Persistent Objects and Classes	5
Accessing Objects	5
Additional Conventions	7
2.USING SMARTEAM COM OBJECTS	9
Creating a Creatable Object	9
Obtaining a Non-Creatable Object	10
Working with Collection Objects	10
Using Scripts	11
Using the SmarTeam Object Model in another Application	12
Add-In Services	14
3.SMARTEAM COM LIBRARIES OVERVIEW	16
SmarTeam Record List Library	16
SmarTeam Engine Library	17
SmarTeam GUI Services Library	17
SmarTeam Utilities Library	17
SmarTeam - Workflow Library	18
SmartMessages Library	18
SmarTeam CAD Interface Library	19
SmarTeam Integration Tools Library	19
SmartIXF Library	20
SmartClientContextService Library	20

SmartFileCatalog Library	20
SmartRecordList Library	21
4.SMARTEAM RECORD LIST LIBRARY	22
General Description	22
Dependencies	22
Overview of Record Lists	22
Overview of Objects	23
SmRecordList Object	23
SmRecord Object	36
SmRecordListHeaders Object	39
SmRecordListHeader Object	40
Grouping Columns in a Record List	40
5.SMARTEAM ENGINE LIBRARY	44
General Description	44
Dependencies	45
Persistent Objects and Classes	45
Overview of Objects	45
SmEngine	46
SmSession Object	50
SmDatabase Object	55
SmConfig Object	58
Metadata Management Objects	64
Persistent Object Management	86
SmQuery Object	104
6.SMARTEAM GUI SERVICES LIBRARY	119
General Description	119
Dependencies	119
GUI Concepts	119
Overview of Objects—ISmCommonGUI	120
The Views Property	122
ISmView	123

Specifying Contents for a Standard View	139
ISmActiveWindow	142
Using ISmView and ISmViewWindow	143
ISmDialogs	145
Basic Dialogs	147
ISmSaveAsDialog.ControlProperties	150
ISmSaveAsDialog.OptionsProperties	152
ISmLocalFilesExplorer	154
ISmSaveAsDialog	158
ISmOpenDialog	163

7.SMARTEAM UTILITIES LIBRARY 169

General Description	169
Dependencies	169
Overview of Objects	170
SmSessionUtil Object	170
Object Functionality	171
File Vault Operations	171
Copied-File Registration	175
Lifecycle Operations	177
SmMiscUtil Object	210
SmConvert and SmSessionConvert Objects	210

8.SMARTEAM - WORKFLOW LIBRARY 214

General Description	214
Overview of Objects	215
SmFlowProcess Object	215
SmFlowChart Object	227
SmFlowSession Object	245
SmWorkflowView Object	258
SmFlowStore Object	260
Overview of the SmartMessage Library Objects	263
SmMessageSession Object	263
SmMessageQueue Object	265
SmMessageStore Object	271

Using the SmarTeam - Workflow Library	273
Writing SmarTeam - Workflow Applications	273
Writing Run-Time Scripts	276
 9.SMARTEAM CAD INTERFACE LIBRARY	 294
General Description	294
Dependencies	296
Overview of Objects	297
SmCADInterface Object	297
 10.SMINTEGRATIONTOOL LIBRARY	 318
Introduction	318
ISmIntegrationStore	320
ISmSpecificIntegrationStore	323
ISmCadFileTypes	326
ISmCadFileType	327
ISmManagedClasses	328
ISmPropertyGroupTypes	332
ISmPropertyGroupType	334
ISmPropertyGroups	336
ISmGroupProperties	340
ISmClassesMappings	345
ISmIntegrationGUIStore	349
ISmPropertiesGroupsGUIService	350
 11.SMARTIXF LIBRARY	 354
Introduction	354
Naming Conventions	354
Overview of Objects	355
ISmIxfSchema	356
ISmIxfClassesBehaviors	359
ISmIxfClasses	363
ISmIxfDomainBehaviors	371
ISmIxfInfo	373

Common Tasks	376
SmlxfInitializationData	380
SmlxfWriter	382
ISmlxfDataWriter	385
ISmlxfSchema	391
Common Tasks	391
SmlxfReader	395
ISmlxfDataReader	396
ISmlxfUnderstoodInfoltems	398
ISmlxfSchema	399
Common Tasks	399
Reading and Writing an External Schema	401
SmlxfExternalSchemaWriter	401
SmlxfExternalSchemaReader	401
ISmlxfStdHelper	402
Standard Behaviors	402
ISmlxfSchemaHelper	403
ISmlxfWriterHelper	410
ISmlxfReaderHelper	434
An IXF Messaging Application	446
Messaging Format	446
Class Behaviors	447
Domain Behaviors	447
Connectivity of Objects	448
Implementing the Application	451
Creating the Schema	451
Writing the Data	460
Reading the Data	467
Executing the Application	470
12.SMARTEAM CLIENT LIBRARIES OVERVIEW	473
SmartClientContext Library	474
ISmClientContext	474
SmartClientContextService Library	475

ISmClientContextService	475
SmartClientServices Library	476
ISmClientServices	476
ISmClientDictionary	477
ISmDictionaryGroup	477
ISmDictionaryProperty	478
SmartClientConfiguration Library	479
ISmClientConfiguration	479
ISmConfigurationValueList	480
SmartInet Library	481
IHttpConnection	481
IHttpContext	481
IHttpUtils	481
SmartFileCatalog Library	483
SmartRecordList Library	483
SmartIntegrationServices Library	483
SmartGUIServices Library	483
SmartEmbeddedScripts Library	484
 13.SMARTFILECATALOG LIBRARY	 485
General Description	485
Dependencies	485
Overview of File Catalog Library	486
File Catalog Object Organization	486
File Catalog in a Shared Workspace	488
File Catalog with Private Files	489
Relation to SmarTeam Processes	490
Overview of Objects—ISmFileCatalog	491
ISmFiles	495
ISmFile	496
ISmFileIdentifiers	507
ISmFileIdentifier	509
ISmFolders	511
ISmFolder	511

ISmWorkspaces	515
ISmWorkspace	516
ISmResultItems	517
ISmResultItem	519
ISmRetrieveFilter	521
Common Tasks	522
14.SMARTRECORDLIST LIBRARY	533
General Description	533
Dependencies	533
Overview of Record List Objects	533
IMutableRecordList	534
IMutableColumns	536
IMutableColumn	537
IMutableRecord	537
IRecordList	539
IColumns	541
IColumn	542
IRecord	542
IRecordListIterator	543
IRecordListUtils	543
IRecordsFactory	544
Events	544
IColumnsChangeEvent	545
IColumnsChangeListener	545
IRecordChangeEvent	545
IRecordChangeListener	546
IRecordListChangeEvent	546
IRecordListChangeListener	546
IRecordListValueChangeEvent	547
IRecordListValueChangeListener	547
Common Tasks	548
APPENDIX A - TIPS FOR WRITING SCRIPTS	551
CAD Integration	556

APPENDIX B - SMARTEAM ADD-IN SERVICES	557
APPENDIX C - WRITING SERVER APPLICATIONS	559
Requirements	559
Guidelines	559
APPENDIX D - SMARTEAM INTEGRATION AND INTEGRATION LINK BEHAVIORS	561
SmarTeam Integration_Behaviors	561
SmarTeam Integration_Link_Behaviors	562

PART I

INTRODUCTION TO SMARTTEAM API

1. Introduction

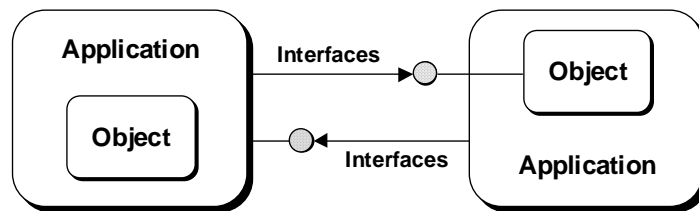
COM is a platform-independent, distributed, object-oriented system for creating software components that can interact with each other.

COM offers the following capabilities:

- Plug-in ability—adding new accessories into existing applications does not require rebuilding the application
- Ability to interact with other objects independently of the working environment
- Accessibility within a single process, in other processes, or from a remote machine
- Standardized object-oriented programming, including:
 - Type of objects
 - Standard methods
 - Naming conventions
 - Encapsulation.

Data associated with a COM object is manipulated through its interfaces. An interface is a class whose members are defined but not implemented.

An interface implementation is associated with an object when an instance of that object is created, as shown in the following schematic:



What is the SmarTeam Object Model?

The **SmarTeam** Object Model provides programmatic access to the functionality of the **SmarTeam** family of products. Using the **SmarTeam** Object Model, users and developers can customize and enhance **SmarTeam - Editor**, as well as build custom solutions that take advantage of the advanced capabilities of the **SmarTeam** engine.

The **SmarTeam** Object Model is exposed as a collection of COM objects and interfaces, and provides:

- Language-independence
- Standard programming paradigms and naming conventions
- Flexibility.

Language-Independence

The **SmarTeam** Object Model can be used from any modern development tool.

Examples of such tools include:

- The Microsoft Visual Studio family of tools, including Visual Basic, Visual C++ and others
- VBScript and Jscript
- Borland Delphi
- Java
- Any other COM-aware tool

Standard Programming Paradigms and Naming Conventions

The **SmarTeam** Object Model is exposed as Automation objects, and uses the same naming conventions as the Microsoft Office Object Models.

Flexibility

The **SmarTeam** Object Model is compatible with Distributed COM (DCOM), and can therefore be accessed from within the same process (single executable), from another process, or from another machine on the network.

Basic Libraries and Service Libraries

The **SmarTeam** Object Model consists of a core API, which provides the basic functionality common to all **SmarTeam** products, as well as an extendable collection of Add-in Services, which provides more specialized functionality.

The core API provides functionality in the following areas:

- Database access
- Working with **SmarTeam** objects
- Queries
- Data structures

The built-in services provide functionality in a number of areas, including, for example:

- Workflow
- User interface
- Integration

The **SmarTeam** Object Model includes basic libraries and add-in libraries, as follows:

- Basic libraries consist of the **SmarTeam** Record List library and **SmarTeam** Engine library.
- Add-in services are located in service libraries, such as the **SmarTeam - Workflow** library. In order to have access to service libraries, the programmer must invoke the appropriate service.

Engine and Session Objects

Through the **SmEngine** and **SmSession** objects, the **SmarTeam** Object Model can support several users working at the same time on the same or different databases.

SmEngine is a creatable object that is a root of the **SmarTeam** Object Model, and provides access to all other **SmarTeam** Object Model objects. The **SmEngine** object has a limited set of methods that mainly serve for creating sessions, manipulating configured databases, and changing the configuration.

The programmer can create several sessions under the **SmEngine** object. Each session enables the programmer to open an individual connection to the database for a specific user. Each session is represented by an **SmSession** object, which usually represents a single user.

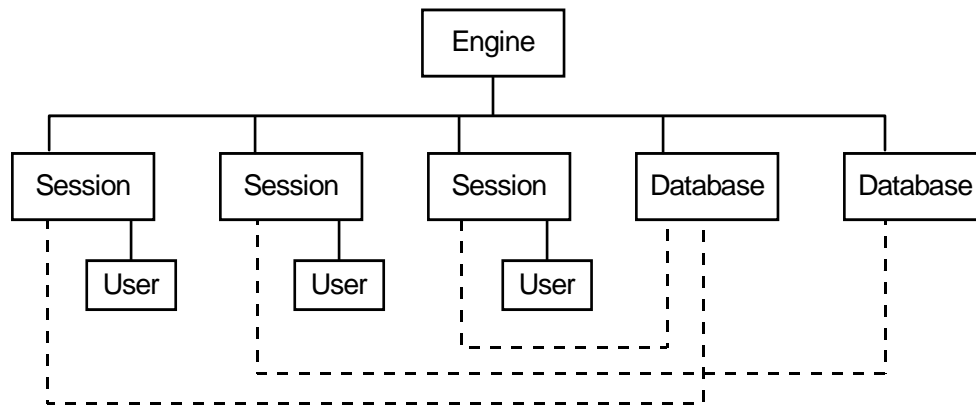
Each session enables access to the **MetaInfo** object, which contains information about the structure of the database opened under the session and to the **UserMetaInfo** object, which contains information about the currently logged-on user for the session. All major objects in the **SmarTeam** Object Model operate within the scope of the session.

The **SmEngine** object keeps a list of all currently open sessions and a list of all databases opened during the sessions.

Through the **SmEngine** and **SmSession** objects, **SmarTeam - Editor** enables the following working flexibility:

- Multiple users
- Multiple databases
- Multiple connections to each database
- Multiple concurrent sessions, each one associated with a specific user and database connection

The following schematic illustrates the multi-user, multi-session and multi-database working flexibility enabled by the **SmEngine** and **SmSession** objects.



Persistent Objects and Classes

The terms Persistent Objects and Persistent Classes are used in connection with the **SmarTeam** Object Model to describe specific objects and classes that are held in the **SmarTeam** database.

See [Chapter 5 - SmarTeam Engine Library](#), for details of Persistent Objects and Persistent Classes.

Accessing Objects

Most of the objects used in the **SmarTeam** Object Model cannot be created directly. Instead, they are accessed and created using properties and methods of other objects in the model. Some **SmarTeam** objects are exceptions to this rule, and can be created externally.

Creatable objects in the **SmarTeam** object model include:

- The **SmEngine** object. When writing scripts from within **SmarTeam - Editor** itself, the **SmEngine** object is always available. However, when creating custom solutions using the **SmarTeam** Object Model outside of **SmarTeam - Editor** itself, the **SmEngine** must be created.
- The **SmRecordList** and **SmRecord** object. This is a stand-alone object, which can be used as a general purpose versatile container, and can be created manually.
- The **SmConfig** object. This object can be used as a stand-alone object that provides convenient access to **SmarTeam - Editor** configuration information. The **SmEngine** and **SmSession** objects contain references to their own **SmConfig** object. Using the **SmConfig** object referenced by **SmEngine** or **SmSession** enables access to application-dependent or user-dependent configuration information.
- The **SmStrings** object. This object provides operations for manipulating string collections.

To create one of these objects, you can use the facilities provided by your development tool to create a COM object. For example, in Visual Basic, this would be the `CreateObject` function, while in Visual C++, you need to use the API function `CoCreateInstance`.

Running the Method ObjectProfile in C++

When performing an Import operation of "smapplic.tlb", you must add `rename ("EOF","EOFX")` and `rename ("LoadLibrary","LoadLibraryX")` by doing the following:

```
#import "smapplic.tlb" no_namespace named_guids
rename("EOF","EOFX") rename ("LoadLibrary","LoadLibraryX")

before #import "smutils" you need to import #import "smartinternal.tlb"
no_namespace named_guids.
```

SmarTeam COM Naming Conventions

The **SmarTeam** Object Model applies the following naming conventions:

- A **SmarTeam** COM object is identified by the prefix **Sm**.
- A **SmarTeam** object interface is identified by the prefix **ISm**.
- An **Enum** variable, used to define two or more constant options, is designated with the suffix **Enum**; for example, **ClassTypeEnum** identifies alternatives relating to **ClassType** object functionality.
- The names of the constant options of an **Enum** variable have a prefix, which is usually a concatenation of two or three capital letters of the variable name. For example, the constant options of the **ClassTypeEnum** are identified by the prefix **ct**, as in **ctComplexLink**, **ctHierLink**.

Additional Conventions

The following conditions also apply:

- A **SmarTeam** COM creatable object can be created using the facilities provided by your development tool to create a COM object.
- A **SmarTeam** COM non-creatable object is obtained through the specific method of another object.
- Exceptions are used to indicate a COM failure condition. This is used in place of API return values.

SmarTeam Object Model Programmer's Guide

2. Using SmarTeam COM Objects

The **SmarTeam** COM Object Model facilitates access to and manipulation of **SmarTeam** data and utilities.

A SmarTeam COM object can be one of the following:

- A creatable object that is independently created, using the facilities provided by the development tool or the COM CoCreateInstance API function.
- A non-creatable object, obtained via method calls of other objects.

Creating a Creatable Object

Each COM class has two identifiers: A Class Identifier (CLSID), which is a globally unique 128-bit numeric identifier, and a Programmatic Identifier (PROGID), which is a more readable, textual name for the class.

To create an externally creatable object, do one of the following:

- Use the PROGID to identify the class at run-time:

```
Set SmEngine = CreateObject("SmApplic.SmEngine")
```

- Include a reference to the corresponding **SmarTeam** Type Library in your project, and use the following syntax:

```
Set SmEngine = New SmApplic.SmEngine
```

When possible, use the second method, since it is more efficient and results in faster creation time.

Note: In most cases you will not be able to work properly with the object without calling the init method, as follows:

```
SmEngine.init "Smtteam32"
```

Obtaining a Non-Creatable Object

A non-creatable object is obtained via method calls of other objects, as follows:

```
Set SmSession = SmEngine.CreateSession(...)
```

Working with Collection Objects

Collections are objects that represent sets of objects or variants over which users can iterate. They are useful in expressing the concept of "sets of things that should be grouped together".

For example, the object **SmRecordListHeaders** represents a collection of **SmRecordListHeader** objects that are grouped in one **SmRecordlist**.

Collection objects always includes the following properties and methods:

- A **Count** property, which returns the number of items in a collection
- An **Item (Index)** property which returns a specific item of the collection **Index** – the key that uniquely identifies the item. **Index** can be type Integer or OleVariant.

If the collection is searchable, it will include an **IndexOf** method, which returns the index of an item in a collection.

If the collection can be modified, it will also include the **Add** and **Remove** methods, allowing the users of a collection object to add or remove items.

Some collections can include:

- An **ItemByName (Name)** method that returns an item of the collection specified by the string name of the item
- An **ItemByIndex (Index)** method that returns an item of the collection specified by integer index.

In addition to these functions and properties, a collection object may expose additional functions and properties, which are specific to its functionality.

Using Scripts

A script is program code that can be executed in the **SmarTeam** system, usually in response to an event.

Using Scripts, the user can customize and enhance the **SmarTeam** family of products.

Examples of scripts include setting up a workflow process, or generating email notification each time a new project is created.

Scripts can be attached to a variety of **SmarTeam** events and executed in a number of ways. Scripts can be attached either to system operations or to specified **SmarTeam** events.

Script argument structures can differ from one event to another. The script programmer should be familiar with the script interface at the particular event hook before beginning to write code.

The **SmarTeam** API provides the SmartConstants object, which allows developers writing in scripting languages, such as VBScript and JScript, to use constants (enumerations) without using their actual numeric value, or having to redeclare them in their own code (see [Appendix A](#) for more details).

For more information about writing scripts, refer to [Appendix A](#).

Using the SmarTeam Object Model in another Application

Unlike **SmarTeam** scripts, if you wish to use the **SmarTeam** Object Model from another application, the **SmEngine** and **SmSession** objects are not automatically available.

To initialize them correctly, follow the steps below:

1. Create and initialize the SmEngine object.
2. Create the SmSession object.
3. Open a database connection to a specific database via an SmSession method (.OpenDatabaseConnection), thereby creating SmDatabase and SmDatabaseConnection objects.
4. Log in via a SmSession method (.UserLogin). You can now work with all other objects.
5. At the end of the application, use method Engine.Terminate.

The above procedure is illustrated in the following example:

```
Sub main()  
  
    Dim Engine As SmApplic.SmEngine  
  
    Dim Session As SmApplic.SmSession  
  
    ' Create SmarTeam Engine object  
  
    Set Engine = New SmApplic.SmEngine  
  
    ' Initiate the Engine object  
  
    Engine.Init "SmTeam32"  
  
    ' Create a session object  
  
    Set Session = Engine.CreateSession("DemoApplication", "SmTeam32")  
  
    ' Open Database connection  
  
    Session.OpenDatabaseConnection "SmDemo", "<password>", true  
  
    ' Log in as "joe"  
  
    Session.UserLogin "joe", ""  
  
    ...
```


<Application Body>

...

Engine.Terminate

End Sub

Add-In Services

Certain services are not created automatically when the session is created. If the associated functionality is required, the user must obtain the service from the session, using the ProgID of the appropriate service library. To obtain the service library, use the **GetService** method of the session, as illustrated in the following example:

```
Sub main()  
  
    Dim Engine As SmApplic.SmEngine  
  
    Dim Session As SmApplica.SmSession  
  
    Dim GUIServices As SmGUIsrv.SmCommonGUI  
  
    'Create SmarTeam engine  
  
    Set Engine = New SmApplic.SmEngine  
  
    ' Initialize Engine object  
  
    Engine.Init "SmTeam32"  
  
    ' Create a session object  
  
    Set Session = Engine.CreateSession("DemoApplic", "SmTeam32"  
  
    ' Open database connection  
  
    Session.OpenDatabaseConnection "SmDemo", "<password>", False  
  
    ' Initialize GUI services object by ProgId SmGUIsrv.SmCommonGUI  
  
    Set GUIServices = Session.GetService("SmGUIsrv.SmCommonGUI")  
  
    ' Open Login window  
  
    GUIServices.Dialogs.ExecuteLogin  
  
    ...  
  
    <Application Body>  
  
    ...  
  
    Engine.Terminate  
  
End Sub
```

For a list of SmarTeam Add-In services, refer to [Appendix B](#).

PART II

SMARTTEAM COM LIBRARIES

3. SmarTeam COM Libraries Overview

This chapter contains a brief overview of the **SmarTeam** COM libraries described in this document.

- **SmarTeam** Record List Library
- **SmarTeam** Engine Library
- **SmarTeam** GUI Services Library
- **SmarTeam** Utilities Library
- **SmarTeam** - Workflow Library
- **SmartMessages** Library
- **SmarTeam** CAD Interface Library
- **SmarTeam** Integration Tools Library
- **SmartIXF** Library

SmarTeam Record List Library

The **SmarTeam** Record List library comprises the set of basic data structure objects that serve as containers for **SmarTeam** data.

The objects in the **SmarTeam** Record List library are expandable, dynamically allocated data structures. The **SmarTeam** Record List library enables you to define new structures on an ad hoc basis, "on the fly".

See [SmarTeam Record List Library](#) for further details.

SmarTeam Engine Library

The **SmarTeam** Engine library contains objects that perform the following basic and advanced **SmarTeam - Editor** functionality:

- Create and initialize the **SmarTeam** system connection to the database, and manage all accesses to the **SmarTeam** Engine and database
- Manage the **SmarTeam** data model and retrieve information about the **SmarTeam** data model
- Manage all **SmarTeam** persistent objects
- Manage the various types of **SmarTeam** queries.

See [SmarTeam Engine Library](#) for further details.

SmarTeam GUI Services Library

The **SmarTeam** GUI Services library comprises objects that enable the following basic functionality:

- Create new GUI components
- Retrieve information about existing GUI components
- Create and display **SmarTeam** views
- Perform other special functionality
- Display various SmarTeam windows and dialogs.

See [SmarTeam GUI Services Library](#) for further details.

SmarTeam Utilities Library

The **SmarTeam** Utilities library comprises objects that enable the following functionality:

- Format conversions
- Mask creation and attribute definition
- Life cycle methods
- Other miscellaneous methods.

See [SmarTeam Utilities Library](#) for further details.

SmarTeam - Workflow Library

The **SmarTeam - Workflow** library comprises objects that enable you to initiate and promote workflow processes. These processes include nodes, users, tasks, to do lists, and associated Workflow functionality.

The **SmarTeam - Workflow** library includes the following functionality:

- Creation and retrieval of objects relating to flowchart processes
- Flowchart design and retrieval, flowchart process, and flow engine functionality
- Creation of flowcharts, including the physical appearance of flowcharts
- Definition of flowchart nodes, including policy, users and/or physical attributes, such as location and size
- Linking of scripts to node events, for example, before send, after send, and open
- Definition of node tasks of various types, manual, operation and script
- Definition of automatic and non-automatic node tasks.

See [SmarTeam - Workflow Library](#) for further details.

SmartMessages Library

The **SmartMessages** library comprises objects, methods and properties that enable you to compose and send **SmarTeam** messages and Internet e-mail messages.

See [SmarTeam - Workflow Library](#) for further details.

SmarTeam CAD Interface Library

The **SmarTeam** CAD Interface library helps you integrate **SmarTeam - Editor** with various CAD systems. The **SmarTeam** CAD Interface library works with the files in the CAD system, saving information about the CAD files in the **SmarTeam** database.

The **SmarTeam** CAD Interface library comprises objects that enable you to:

- Save and update documents and compositions (trees) of the documents in the **SmarTeam** database
- Retrieve document meta-information from the **SmarTeam** database by file name
- Update various blocks of the drawing with meta-information
- Perform life cycle operations
- Other related operations.

See [Introduction](#) for further details.

SmarTeam Integration Tools Library

The **SmarTeam** Integration Tools library comprises objects that enable you to define relations between the **SmarTeam** data model and components of the CAD system. This library contains methods that enable you to:

- Define default class and File Type for the Integration Behaviors
- Set up mappings between CAD file property fields and SmarTeam class attributes in an integration

See [SmIntegrationTool Library](#) for further details.

SmartIXF Library

iXF is an XML-based format for describing a collection of objects and associated files that conform to a specific data model. iXF is used to exchange this data between systems.

The **SmartIXF** library comprises objects that enable you to perform the following functions:

- Generating an iXF schema
- Processing an iXF schema
- Generating an iXF Archive File
- Processing an iXF Archive File

See [SmartIXF Library](#) for further details.

SmartClientContextService Library

This library provides access to the various Client libraries.

See [SmarTeam Client Libraries Overview](#) for further details.

SmartFileCatalog Library

The **SmarTeam** File Catalog library comprises objects that enable the following functionality:

- Client-side file management for running SmarTeam in all possible configurations
- Provides API support for integrations, life-cycle operations, SmarTeam File Explorer, and collaborative design
- Folder-based structure management for file storage on client's machine or in network location
- Provides a mechanism for accessing and updating file object attributes

See [SmarTeam Client Libraries Overview](#) for further details.

SmartRecordList Library

The SmartRecordList library comprises objects that enable the following functionality:

Allows a client to work with record list data objects that are similar to those in the SmarTeam API

See [SmartFileCatalog Library](#) for further details.

4. SmarTeam Record List Library

General Description

The **SmarTeam** Record List library provides the **SmRecordList** object, a versatile data structure that serves as a generic container for **SmarTeam** data.

The **SmRecordList** object is a dynamically allocated, expandable object that allows you to define new data structures on an ad hoc basis.

Dependencies

The **SmarTeam** Record List library has no dependent COM libraries.

Overview of Record Lists

A **SmRecordList** object is a flexible, matrix-like data structure consisting of a variable number of records and attributes. Each node in the matrix contains a value.

A column comprises a header and associated value nodes for each column. The header contains identifying information about the column.

An **SmRecord** object represents a row or a subset of a row of an **SmRecordList**, and can contain one value node for each column.

The **SmarTeam** Record List library includes a grouping feature that enables you to access specific parts of a record list object according to predefined groups. By using this feature, a subset of columns or a subset of a record can be represented and you can use the same header name in two different groups.

Overview of Objects

The **SmRecordList** object is the creatable object that defines the basic data structures used by **SmarTeam - Editor**, and serves as a dynamic container for **SmarTeam** data.

The following diagram shows the major objects in this library:

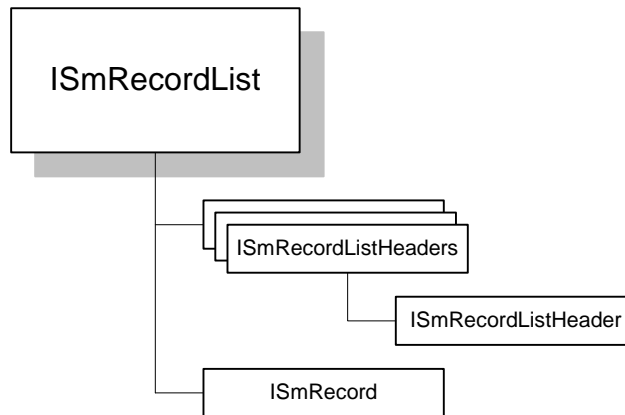


Figure 4-1 *ISmRecordList Object Diagram*

SmRecordList Object

Unlike static arrays or tables, the **SmRecordList** object is a variable data structure that is:

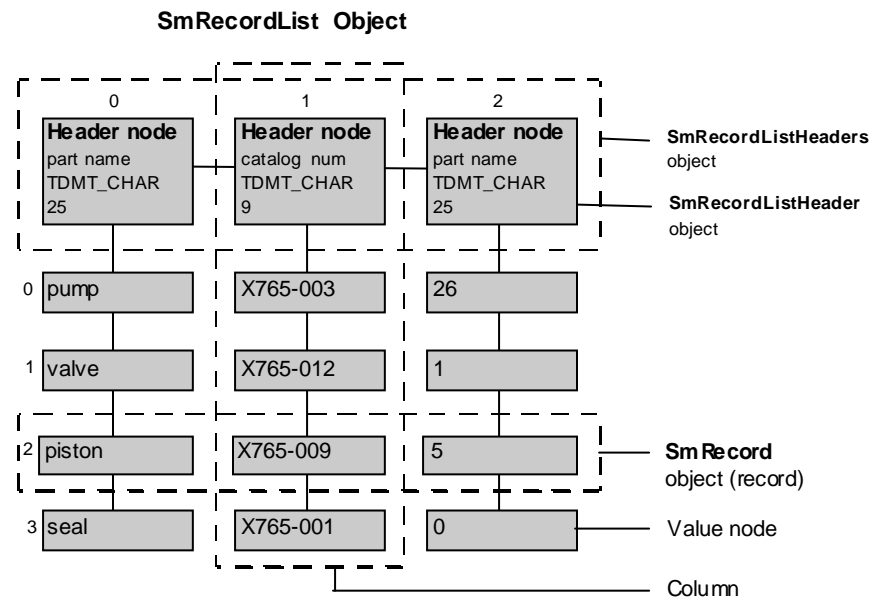
- Dynamically allocated, to accommodate the attributes of all future data objects with maximum flexibility
- Expandable, to hold as much information as necessary
- Generic, to accommodate any format.

This enables you to define new **SmarTeam** structures, as you need them.

The **SmRecordList** object holds information in the form of records of values where each node of a record can have a different value type. Nodes in the RecordList with the same record index represent the same data type.

The **SmRecordList** object can be thought of as a matrix whose first row is a header row comprising header nodes and whose subsequent rows are records comprising value nodes. Thus, the first element of each column of this matrix is a header node and subsequent column elements are value nodes for each record corresponding to the header.

The example below illustrates a sample **SmRecordList** object structure:



Each header node of a column contains the following information, deriving from the **SmRecordHeader** object:

- **Name**: Unique string identifying the column.
- **Type**: Identifies the type of data contained in the column, as one of the constants in the Enum **SmDataTypesEnum**.
- **Size**: Size (in bytes) for each value node, if applicable.

The header nodes are uniquely indexed from **0** to **n-1** (**n** being the current number of columns in the **SmRecordList** object). The value nodes in each column are similarly indexed from **0** to **n-1** (**n** being the current number of value nodes in the column).

Properties

The ISmRecordList object has the following properties

Property	Description
CapacityIncrement	Retrieves or sets the secondary allocation size. When "CapacityIncrement" is X, there will be X new nodes added to the header when memory needs to be increased.
DynamicValueSize	Returns the actual Integer value size for a node specified by header name and record index.
DynamicValueSizeByIndex	Returns the actual Integer value size for a node specified by header index and record index.
FormattedValueAsStringByIndex	Gets or sets a value for a node specified by header index and record index, where the value is represented as String, in specified Format.
Group	Gets the name of the group in this SmRecordList with a specified group index. Group indexes start with 0.
GroupCount	Retrieves the number of groups in this SmRecordList.
GroupDynamicValueSize	Returns the actual value size for a node specified by group name, header name, and record index.
GroupDynamicValueSizeByIndex	Returns the actual value size for a node specified by group name, relative header index and record index.
GroupHeaders	Returns a collection SmRecordListHeaders for a specified group in the SmRecordList.
GroupValue	Gets or sets a node value for a header in a group, where the node is specified by group name, header name and record index.
GroupValueByIndex	Gets or sets a node value for a header in a group, where the node is specified by group name, header index relative to the group and record index.
HeaderCount	Returns the number of headers.
HeaderIndex	Returns the header index for a header specified by header name.
HeaderName	Returns the header name for a header specified by header index.
Headers	Returns a collection SmRecordListHeaders for this SmRecordList.
InitialCapacity	Retrieves or sets the initial allocation size. When "InitialCapacity" is X, the new header will have X nodes.
MaxValue	Retrieves the maximum value within a collection of nodes in a specified header.
MinValue	Retrieves the minimum value within a collection of nodes in a specified header.
RecordCount	Returns the number of records.
Value	Gets or sets a value for a node specified by header name and record index.

Property	Description
ValueAs[Type]	Gets or sets a value for a node specified by header name and record index, where the value is represented as [Type], one of SmDataTypesEnum.
ValueAs[Type]By Index	Gets or sets a value for a node specified by header index and record index, where the value is represented as [Type], one of SmDataTypesEnum.
ValueByIndex	Gets or sets a value for a node specified by header index and record index.
ValueSize	Returns the header Integer Value Size for a header specified by name.
ValueSize2	Returns the header Long Value Size for a header specified by name.
ValueSizeByIndex	Returns the header Integer Value Size for a header specified by index.
ValueSizeByIndex2	Returns the header Long Value Size for a header specified by index.
ValueType	Returns the header value type for a header specified by name.
ValueTypeByIndex	Returns the header value type for a header specified by header index.

Methods

The ISmRecordList object has the following Methods

Method	Description
Creating SmRecordList Objects	
CreateFromRecord	Adds the headers of a source SmRecord to this SmRecordList wherever the headers are different from the existing headers of SmRecordList. The nodes of the added headers are set to nil.
CreateAsCopy	Replaces this SmRecordList with a copy of the source SmRecordList without any records
CloneHeaders	Creates and returns an SmRecordList that has the same headers as this SmRecordList but contains no records.
ClearValues	Deletes all records from the SmRecordList while retaining the headers.
Adding Headers and Records	
AddHeader	Adds a header, defined by its name, type and value size, to this SmRecordList, where the header value size can be Integer.
AddHeader2	Adds a header, defined by its name, type and value size, to the SmRecordList, where the header value size can be Long.
InsertHeader	Inserts a header with specified name, type and value size into SmRecordList, at a specified header index.
AddRecord	Adds an empty record to SmRecordList and returns its index.
Getting and Finding Records in a RecordList	
GetRecord	Creates and returns an SmRecord from a record of this SmRecordList, specified by record index.
GetGroupRecord	Creates and returns an SmRecord of a group of this SmRecordList, specified by group name and record index.
SearchRecord	Searches this SmRecordList for a given SmRecord. If found, returns the record index, otherwise returns -1.
GroupSearchRecord	Searches a group of this SmRecordList for a given SmRecord. If found, returns the record index, otherwise returns -1.
SmRecordExists	Returns True if a given SmRecord, exists in this SmRecordList
RecordExists	Returns True if the record of a SmRecordList, specified by record index, exists in this SmRecordList.
OneFieldFilter	Finds a record which has a given header node value in a SmRecordList. Returns either the record found or the record index.
MultiFieldFilter	Finds records with given header node values in a SmRecordList. Returns a SmRecordList containing either the records found or indexes of the records found.

BinSearch	Searches for records with specified node values in an SmRecordList.
GroupBinSearch	Same as BinSearch, but limited to the headers of a specified group.
Finding a Node of a Header	
GetRecordIndex	Returns the record index of the first header node that matches given value for a header specified by name. Returns -1 if no node is found.
GetRecordIndexByIndex	Returns the record index of the first header node that matches given value for a header specified by index. Header indexes start from 0. Returns -1 if no node is found.
GetRecordIndexStartFromIndex	Same as GetRecordIndexByIndex, with the search starting point specified by 'InitRow'.

Combining Two SmRecordList Objects	
Cat	Appends (concatenates) the records of the specified RecordList to this SmRecordList. Optionally, before executing the method verifies that the headers of both SmRecordLists match.
CatRecord	Appends (concatenates) the specified SmRecord of a source SmRecordList to this SmRecordList.
InsertRecords	Copies a sequence of records from a SmRecordList and inserts them into this SmRecordList at a specified record index, shifting the existing records.
Merge	Adds (merges) to this SmRecordList columns of a specified SmRecordList whose headers do not exist in this SmRecordList.
Sub	Removes all headers in this SmRecordList that also appear as headers in a given SmRecordList.
Copying SmRecordList Objects	
Copy	Replaces this SmRecordList with an exact copy of the specified source SmRecordList.
CopyExternal	When the headers are the same, nodes of this SmRecordList are overwritten with corresponding nodes of the source SmRecordList. Headers and records in the source SmRecordList that do not exist in this SmRecordList are added to it. Information is copied from one record list to another, even if they are in different processes.
Copying SmRecord Objects	
CopyRecord	Copies the specified SmRecord from the source SmRecordList and overwrites the destination SmRecord in this SmRecordList.
CopyRecordByOrder	Copies the specified SmRecord from the source SmRecordList and overwrites the destination SmRecord in this SmRecordList. The nodes are copied according to their sequential order, disregarding the header names.
CopyRecordExt	Copies the specified SmRecord from the source SmRecordList and overwrites the destination SmRecord in this SmRecordList. If SkipDiffTypes is True, values of non-matching elements will not be copied.
CopySmRecord	Similar to CopyRecord except it operates on an isolated SmRecord instead of on a record in an SmRecordList.
CopySmRecordByOrder	Similar to CopyRecordByOrder except it operates on an isolated SmRecord instead of on a record in an SmRecordList.
CopySmRecordExt	Similar to CopyRecordExt except it operates on an isolated SmRecord instead of on a record in an SmRecordList.
CatSmRecord	Appends (concatenates) an SmRecord to this SmRecordList.

Sorting Records	
Sort	Sorts the records in this SmRecordList according to the header names, sort directions and sort order specified in the SortQuery SmRecordList.
Deleting	
Clear	Deletes all headers and nodes in the SmRecordList object.
DeleteElement	Deletes the entire column specified by a header name from the SmRecordList.
DeleteElementBy Index	Deletes the entire column specified by a header index from this SmRecordList. Header indexes start with 0.
DeleteRecord	Deletes the record specified by a record index from this SmRecordList. Record indexes start with 0.

Exchanging	
Exchange	Exchanges two nodes of a header in this SmRecordList object.
ExchangeByIndex	Exchanges two records in this SmRecordList object.
Miscellaneous	
GroupExist	Returns True if a specified group exists in this SmRecordList.
Init	Defines the initial amount of memory and the amount of reallocation memory, both in number of records.
PrintToFile	Prints the nodes of this SmRecordList to a file in a record by record format.
Indexed Searches	
CreateIndex	Creates an index for a header with a specified position.
CreateIndexByName	Creates an index for a header with a specified header name.
IsIndexed	Checks if the column in a Record List with a specified column number is indexed.
IsIndexedByName	Checks if the column in a Record List with a specified column name is indexed.
RemoveIndex	Removes indexing from the column with a specified column number.
RemoveIndexByName	Removes indexing from the column with a specified column na

Example

The following examples finds selected records in a record list, copies them to a new record list and deletes them in the original record list.

```
Dim RecordListA As SmRecList.SmRecordList

Dim RecordListB As SmRecList.SmRecordList

Dim Query As SmRecList.SmRecordList

Dim SRecordIndex As Integer

Dim DRecordIndex As Integer

Dim I As Integer

` Populate record list

Set RecordListA = New SmRecList.SmRecordList

RecordListA.AddHeader "ObjectId", 4, sdtInteger

RecordListA.AddHeader "FileName", 256, sdtChar
```

```
RecordListA.AddHeader "Directory", 256, sdtChar

RecordListA.ValueAsInteger("ObjectId", 0) = 1

RecordListA.Value("FileName", 0) = "File2"

RecordListA.Value("Directory", 0) = "Dir2"

RecordListA.ValueAsInteger("ObjectId", 1) = 3

RecordListA.Value("FileName", 1) = "File1"

RecordListA.Value("Directory", 1) = "Dir1"

RecordListA.ValueAsInteger("ObjectId", 2) = 2

RecordListA.Value("FileName", 2) = "File2"

RecordListA.Value("Directory", 2) = "Dir2"

'Sort it according to FileName

Set Query = New SmRecList.SmRecordList

Query.AddHeader "COL_NAME", 256, sdtChar

Query.AddHeader "TDM_DIRECTION", 4, sdtBoolean

Query.Value("COL_NAME", 0) = "FileName"

Query.ValueAsBoolean("TDM_DIRECTION", 0) = True

'Query.Value("COL_NAME", 1) = "H2"

'Query.ValueAsBoolean("TDM_DIRECTION", 1) = False

RecordListA.Sort Query, 1, 0

'MultiFieldFilter to find records with given Filename and Directory

Set Query = New SmRecList.SmRecordList

Query.AddHeader "FileName", 256, sdtChar

Query.AddHeader "Directory", 256, sdtChar

Query.Value("FileName", 0) = "File2"

Query.Value("Directory", 0) = "Dir2"

' RecordListB receives indexes of found records
```

```

Set RecordListB = RecordListA.MultiFieldFilter(Query, False)

` Make RecordListC to copy found records to

Dim RecordListC As SmRecList.SmRecordList

Set RecordListC = New SmRecList.SmRecordList

RecordListC.CreateAsCopy RecordListA, 1

` Copy found records to RecordListC and delete them from RecordListA

For I = RecordListB.RecordCount - 1 To 0 Step -1

    SRecordIndex = RecordListB.ValueAsSmallInt("SERV_FIELD", I)

    DRecordIndex = RecordListB.RecordCount - 1 - I

    RecordListC.CopyRecord RecordListA, SRecordIndex, DRecordIndex

    RecordListA.DeleteRecord (SRecordIndex)

Next I

```

Indexed Searches of RecordLists

The RecordList library provides the ability to perform indexed searches of a RecordList, where the key used in the search can be specified by the user as a header of the RecordList.

The indexed search ability is provided within the framework of the existing ISmRecordList search methods; you cause a method to use an indexed search instead of a sequential search by setting up indexing for headers of the specific RecordList to be searched.

Methods that can use Indexing

The following ISmRecordList methods use indexing when at least one of the headers used has been set up for indexing:

Method	Remarks
GetRecordIndexByIndex	
GetRecordIndexStartFromIndex	
OneFieldFilter	Uses indexing when the match Record List includes or single value for the header.
MultiFieldFilter	Uses indexing when the match Record List includes or single set of values – one match value for each header
SearchRecord	

GroupSearchRecord	
SmRecordExists	

Setting up Indexing

Using the methods `CreateIndex` or `CreateIndexByName`, the user specifies one or more headers (columns) of a `RecordList` to be searched to be possible keys for a binary search. For each header specified as a key, a separate key array is constructed and sorted in ascending order, where each key carries a reference back to its original record in the `RecordList`. The binary search is carried out on the key array.

Only one key is actually used in a search. In case more than one header of a `RecordList` is specified as a key, the user can determine which key will actually be used by specifying a priority among the keys using `IndexType`, which can have the values:

itUnique - Assign this value to a header when you want the system to use it as a key as a first choice.

itNotUnique - Assign this value to headers that you want the system to use as a second choice.

You do not have to assign indexing to each header of the `RecordList`.

Method of Search according to Indexing Setup

After you have set up indexing for a `Record List` and you then call one of the `ISmRecordList` methods that can use indexing, the behavior of the method depends on which `RecordList` headers you use in the method call.

For example, when you search `SmRecordList` using:

```
SmRecordList.SearchRecord(Query)
```

The `ISmRecord Query` parameter method can contain several headers of the `SmRecordList`, some of which were indexed and some of which were not indexed.

The actual search can be a combination of an indexed search and a sequential search, depending on the headers.

The following table shows the method of search that is used in the SearchRecord method, according to various cases of setting indexing for the SmRecordList headers.

Case	Query contains SmRecordList headers to:			Method of search
	itUnique	itNotUnique	noindexing	
1	x	x	x	1. Performs index search with itUnique header as key. 2. Performs sequential search on record with the same index values.
2		x	x	1. Performs index search with itNotUnique header as key. 2. Performs sequential search on record with the same index values.
3			x	Performs sequential search with noindexing header.

When to use Indexing

Setting up for an indexed search also requires resources. An indexed search is more efficient than a sequential search only when:

- Searching a large RecordList
- Performing multiple searches of a RecordList

Example

The following are examples of multiple searches of a RecordList made more efficient by using predefined keys.

```
Set SortQuery = New SmRecList.SmRecordList
Set QueryByIndex = New SmRecList.SmRecord
Set QueryByFileName = New SmRecList.SmRecord
Set QueryComposite = New SmRecList.SmRecord
QueryByIndex.AddHeader "OBJECT_ID", 4, sdtInteger
QueryComposite.AddHeader "OBJECT_ID", 4, sdtInteger
QueryByFileName.AddHeader "FILE_NAME", 255, sdtChar
QueryComposite.AddHeader "FILE_NAME", 255, sdtChar
```

```
SortQuery.AddHeader "COL_NAME", 30, sdtChar
```

```
'Create keys
```

```
ExampleList.CreateIndexByName "OBJECT_ID", itUnique
```

```
ExampleList.CreateIndexByName "FILE_NAME", itNotUnique
```

```
'Multiple Index search by OBJECT_ID key
```

```
For I = 0 To Cnt - 1
```

```
    QueryByIndex.ValueAsInteger("OBJECT_ID") = I
```

```
    RecIdx = ExampleList.SearchRecord(QueryByIndex) '
```

```
Next
```

```
'Multiple Index search by FILE_NAME key
```

```
For I = 0 To Cnt - 1
```

```
    QueryByFileName.ValueAsString("FILE_NAME") = CStr(I Mod 500)
```

```
    RecIdx = ExampleList.SearchRecord(QueryByFileName)
```

```
Next
```

```
'Multiple Index search by OBJECT_ID and FILE_NAME key
```

'The query in this case will actually use only the OBJECT_ID key since it was assigned a higher priority.

```
For I = 0 To Cnt - 1
```

```
    QueryComposite.ValueAsInteger("OBJECT_ID") = I
```

```
    QueryComposite.ValueAsString("FILE_NAME") = CStr(I Mod 500)
```

```
    RecIdx = ExampleList.SearchRecord(QueryComposite)
```

```
Next
```

SmRecord Object

The **SmRecord** object represents a record, or a row, in the **SmRecordList** data structure. The **SmRecord** object is also used to represent a subset of a row.

Unlike rows in a static table, **SmRecord** objects are not physically part of the **SmRecordList** object. The two objects are logically related, with the **SmRecord** objects being projections of **SmRecordList** objects.

A **SmRecord** object is obtained in two ways:

- By using the **SmRecordList.GetRecord** method
- By accessing the **SmRecord** object directly.

With the first method, the lifetime of a record depends on the lifetime of the **SmRecordList** object. The **SmRecord** becomes invalid after the **SmRecordList** object is destroyed. With the second method, the lifetime of the record is independent of the **SmRecordList** object.

Properties

The ISmRecord object has the following properties

Property	Description
Constants	Gets Smart Constants.
DynamicValueSize	Returns the actual Integer value size for a node specified by header name.
DynamicValueSizeBy Index	Returns the actual Integer value size for a node specified by header index.
FormattedValueAs String	Gets or sets a value for a node specified by header name, where the value is represented as String in specified Format.
FormattedValueAs StringByIndex	Gets or sets a value for a node specified by header index, where the value is represented as String in specified Format.
Index	Returns the appropriate record index.
Constants	Gets Smart Constants.
DynamicValueSize	Returns the actual Integer value size for a node specified by header name.
DynamicValueSizeBy Index	Returns the actual Integer value size for a node specified by header index.
FormattedValueAs String	Gets or sets a value for a node specified by header name, where the value is represented as String in specified Format.
FormattedValueAs StringByIndex	Gets or sets a value for a node specified by header index, where the value is represented as String in specified Format.
Index	Returns the appropriate record index.
Constants	Gets Smart Constants.
DynamicValueSize	Returns the actual Integer value size for a node specified by header name.
DynamicValueSizeBy Index	Returns the actual Integer value size for a node specified by header index.
FormattedValueAs String	Gets or sets a value for a node specified by header name, where the value is represented as String in specified Format.
FormattedValueAs StringByIndex	Gets or sets a value for a node specified by header index, where the value is represented as String in specified Format.
Index	Returns the appropriate record index.
Constants	Gets Smart Constants.
Value	Gets or sets a value for a node specified by header name.
ValueAs[Type]	Gets or sets a value for a node specified by header name, where the value is represented as [Type], one of SmDataTypesEnum.
ValueAs[Type]By Index	Gets or sets a value for a node specified by header index, where the value is represented as [Type], one of SmDataTypesEnum.
ValueByIndex	Gets or sets a value for a node specified by header index.

Methods

The ISmRecord object has the following methods

Method	Description
Adding Headers and Records	
AddHeader	Adds a header, defined by its name, type and value size, to this SmRecord, where the header value size can be Integer.
AddHeader2	Adds a header, defined by its name, type and value size, to this SmRecord, where the header value size can be Long.
Copying SmRecord Objects	
Copy	Replaces this SmRecord with an exact copy of the specified source SmRecord.
CopyEx	Nodes from the Source SmRecord overwrite the corresponding nodes in this SmRecord.
Deleting	
DeleteElement	Deletes the entire column specified by a header name from the SmRecord.
DeleteElementByIndex	Deletes the entire column specified by a header index from the SmRecord.
PrintToFile	Prints the nodes of this SmRecord to a file in a record by record format.
SetNullValues	Sets the all node values of this SmRecord to nil.

SmRecordListHeaders Object

The **SmRecordListHeaders** object is the collection that comprises a number of **SmRecordListHeader** objects, each one defining a header node of a column of the **SmRecordList** or **SmRecord** object.

Methods

The ISmRecordListHeaders object has the following methods

Method	Description
IndexOf	Returns the index of the SmRecordListHeader with a given name.
HeaderExists	Returns True if the SmRecordListHeader with a given name exists.
ItemByName	Returns the SmRecordListHeader with a given name.
ItemByIndex	Given an index, creates and returns an object representing the header.

SmRecordListHeader Object

The **SmRecordListHeader** object represents the header node of a column. It defines the attributes of a specific header in an **SmRecordList** or **SmRecord** object.

Each header node of a column contains the following information:

- **Name:** Unique string identifying the header node
- **Type:** Identifies the type of data contained in the node, as one of the constants in the Enum **SmDataTypesEnum**
- **Size:** Size (in bytes) for each header node.

Properties

The **ISmRecordListHeaders** object has the following properties

Property	Description
Index	Returns the Header index.
Name	Returns the Header name.
ValueSize	Returns an Integer ValueSize for headers that were created with the method ISmRecordList.AddHeader .
ValueSize2	Returns an Long ValueSize for headers that were created with the method ISmRecordList.AddHeader2 .
ValueType	Returns the Header value type.

Grouping Columns in a Record List

The grouping feature enables you to keep data columns with the same name in one record list, by adding a prefix to the name. A group represents the columns in the **SmRecordList** object that have the same prefix in their name.

SmRecordList object column names can be defined in the header node in the following form:

< prefix>.< name> ,

where:

< **prefix** > identifies a selection of columns. For example, the prefix 1 represents the unique Project class ID, and the prefix 8 represents the unique Project Tree class ID.

<name> identifies the actual name of the column which several other selections may have in common, for example, **CLASS_ID** or **OBJECT_ID**.

For example, when the **CLASS_ID** and **OBJECT_ID** attributes are represented in both the Project class and the Project Tree class, the use of a prefix enables keeping the names of the attributes unique within the **SmRecordList** object.

Columns of the **SmRecordList** object can then be usefully filtered into **groups** by means of the prefix. For example, you can use grouping to create a projection that isolates the columns that relate only to the Project class.

If an **SmRecordList** object is grouped, the methods and properties are executed according to the object grouping.

Grouping Nodes in a Record

An **SmRecord** object is similarly divisible into groups and can be represented in a **grouped SmRecord** object. The grouped **SmRecord** object references a subset of the row that includes nodes that have the same prefix, comprising a specific group.

You can access a grouped **SmRecord** that represents the value nodes within a specific group by supplying an index value for the group name, using the **SmRecordList.GetGroupRecord** method.

You can access a specific header in the **SmRecordListHeaders** of the grouped **SmRecord**, using the related header's index or name of the related header. You do not need to supply the group information, as this was supplied when the group was created.

Example

The following example shows how to perform a binary search on a group in the **SmRecordList**.

```
Dim RecordListA As SmRecList.SmRecordList

Dim Query As SmRecList.SmRecordList

Dim IsFound As Boolean

'

'RecordListA -- Set up with two groups, G1 and G2
```

```

'      G1      G2
' -----
'  H1      H2      H1      H2
' -----

'G1.A11  G1.A12  G2.A11  G2.A12
'G1.A21  G1.A22  G2.A21  G2.A22
'G1.A31  G1.A32  G2.A31  G2.A32
'

Set RecordListA = New SmRecList.SmRecordList
RecordListA.AddHeader "G1.H1", 256, sdtChar
RecordListA.AddHeader "G1.H2", 256, sdtChar
RecordListA.AddHeader "G2.H1", 256, sdtChar
RecordListA.AddHeader "G2.H2", 256, sdtChar
RecordListA.GroupValue("G1", "H1", 0) = "G1.A11"
RecordListA.GroupValue("G1", "H2", 0) = "G1.A12"
RecordListA.GroupValue("G2", "H1", 0) = "G2.A11"
RecordListA.GroupValue("G2", "H2", 0) = "G2.A12"
RecordListA.GroupValue("G1", "H1", 1) = "G1.A11"
RecordListA.GroupValue("G1", "H2", 1) = "G1.A22"
RecordListA.GroupValue("G2", "H1", 1) = "G2.A11"
RecordListA.GroupValue("G2", "H2", 1) = "G2.A22"
RecordListA.GroupValue("G1", "H1", 2) = "G1.A31"
RecordListA.GroupValue("G1", "H2", 2) = "G1.A32"
RecordListA.GroupValue("G2", "H1", 2) = "G2.A31"
RecordListA.GroupValue("G2", "H2", 2) = "G2.A32"

'BinSearch

Set Query = New SmRecList.SmRecordList

```

```

\
\  H1      H2
\  -----
\  G2.A11  G2.A12
\

Query.AddHeader "H1", 256, sdtChar
Query.AddHeader "H2", 256, sdtChar
Query.Value("H1", 0) = "G2.A11"
Query.Value("H2", 0) = "G2.A12"

MsgBox CStr(RecordListA.GroupBinSearch("G2", Query, IsFound)) '= 0
MsgBox IsFound                                     ' = True
MsgBox CStr(RecordListA.GroupBinSearch("G1", Query, IsFound)) '= -1
MsgBox IsFound                                     '= False

'Example
'
'      G1      G2
'-----
'  H1      H2      H1      H2
'----- .GroupBinSearch("G2", Query, IsFound) = 0
'G1.A11  G1.A12  G2.A11  G2.A12
'G1.A21  G1.A22  G2.A21  G2.A22
'G1.A31  G1.A32  G2.A31  G2.A32

```

5. SmarTeam Engine Library

General Description

The **SmarTeam** Engine library provides the basic functionality common to all applications using the **SmarTeam** Object Model. Among the features this library provides are:

- Create and manage sessions with the **SmarTeam** engine—support for multiple users, each one associated with an **SmSession** object
- Establish and manage connections to the **SmarTeam** databases—support for multiple databases
- Retrieve and manipulate the Meta-information, which describes the **SmarTeam** data model
- Retrieve, update and delete Persistent Objects
- Manage the lifetime of **SmarTeam** objects
- Creating and running **SmarTeam** queries
- Support for multi-threaded applications.

The **SmarTeam** Engine Library objects are explained in this chapter under the following headings:

- [SmEngine](#) and [SmSession Object](#), page [46](#), describes the **SmEngine** and the **SmEngine Session** objects, which provide access to the rest of the **SmarTeam** Object Model
- [Note: When using](#) the function `Session.Config.ReadSection`, the `Key` is always returned in lowercase.
- Metadata Management Objects, page [66](#), describes the objects that contain information relating to the **SmarTeam** data model
- [Persistent Object Management](#), page [86](#), describes the objects that enable the creation, update and deletion of **SmarTeam** Persistent Objects, and the retrieval of information about these objects
- [SmQuery Object](#), page [104](#), describes the facilities provided by the **SmarTeam** Object Model to create high-level and low-level queries, and to retrieve the results of such queries.

Dependencies

The **SmarTeam** Engine library is used by all libraries except the **SmarTeam** Record List Library.

The **SmarTeam** Engine library uses the **SmarTeam** Record List Library.

Persistent Objects and Classes

The terms Persistent Objects and Persistent Classes are used in connection with the **SmarTeam** Object Model to describe specific objects and classes that are stored in the **SmarTeam** database.

The **SmObject** and **SmClass** objects, described on pages 86 and 66 respectively, represent these objects and classes.

Overview of Objects

The main objects, which provide access to the **SmarTeam** object model are:

- **SmEngine**, described on page [46](#)
- **SmSession**, described on page [50](#)

Other important objects are as follows:

- **SmDatabase**, described on page [55](#), provides access to **SmarTeam** database functionality
- **SmConfig**, described on page [58](#), provides access to **SmarTeam** system configuration
- **SmMetaInfo**, described on page [66](#), provides access to **SmarTeam** data model functionality
- **SmObjectStore**, described on page [86](#), provides access to **SmarTeam** Persistent Objects
- **SmObject**, described on page [86](#), represents a single persistent object in the **SmarTeam** database
- **SmQueryDefinition**, described on page [104](#), is used to define attribute-based queries.

SmEngine

The **SmEngine** object is the highest level object for the **SmarTeam** Object Model.

The SmEngine and its major components is shown in the following object diagram.

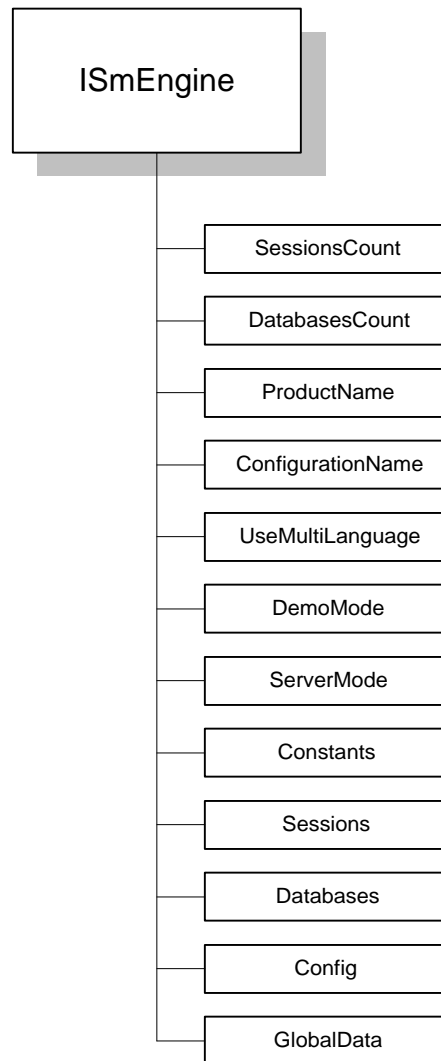
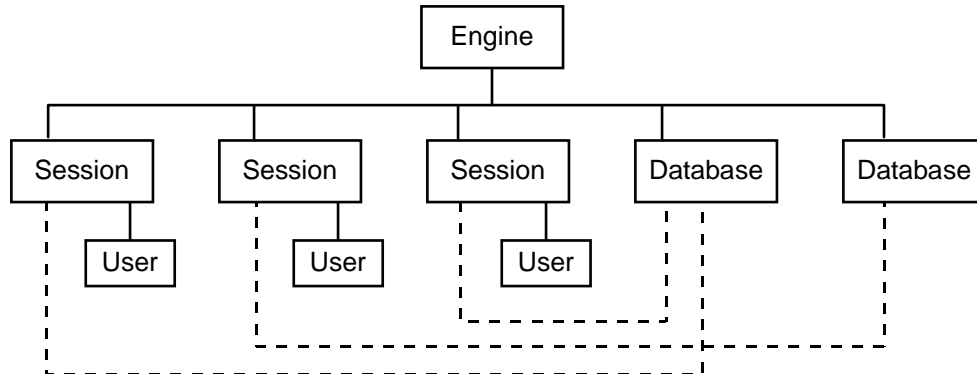


Figure 5-1 SmEngine Object Diagram

The following schematic shows the relationship between the **SmEngine** object and the other objects in the system.



Properties

The ISmEngine object has the following properties

Property	Description
Sessions	
Sessions	Returns one of the collection of the currently open sessions, as ISmSession
SessionsCount	Returns the number of open sessions.
Databases	
Databases	Returns one of the collection of open Databases as listed in the SmarTeam configuration, as ISmDatabase
DatabasesCount	Returns the number of defined databases
Configuration	
Config	Returns an SmConfig object, which represents the current SmarTeam configuration, as ISmConfig
ProductName	Returns or sets the name of the product. Defaults to SmarTeam.
ConfigurationName	Returns or sets the SmEngine configuration INI file name, for example, SmarTeam32.INI. The default location for this configuration file is \SmarTeam\LocalConfig\ SmarTeam32.INI
UseMultiLanguage	Set to True if the menus can be translated into a different language such as French or German, otherwise set to False.
Operation	
DemoMode	If True, a license was not detected and the system automatically entered demo mode, which has some operating restrictions.

Property	Description
ServerMode	Set to True to cause the application to run in Server Mode. In Server Mode, errors are not displayed on the screen, and server-side hooks are used.
Data Access	
Constants	Accesses SmarTeam Enum constants from the SmApplic Library constant name
GlobalData	Provides access to an Engine-wide shared storage area, which can be used to store and retrieve data throughout the life of the application. You can use the GlobalData object to transfer these items between scripts. Data is stored as a VariantList object.

Methods

The ISmEngine object has the following methods

Method	Description
Sessions	
CreateSession	Creates a new SmSession object.
FindSession	Searches for a specified session in the currently connected sessions list.
FindSessionByDatabase Alias	Searches for a session connected to a database specified by its alias.
Databases	
FindDatabase	Searches for a specified database in the defined databases list.
FindDatabaseByReplica Identifier	Returns the Database object corresponding to the specified Replica Identifier.
ReloadDatabasesList	Reloads the list of configured databases.
GetDatabaseAliasesList	Gets a Record List that includes, at least, the password and database name for each database available.
SaveDatabasesList	Saves current database connection information.
Configuration	
Init	Initializes the internal arguments of SmEngine according to data in the INI file. Called directly after creating the SmEngine.
SetInitFlags	Sets the initialization flags. Internal use only.
GetInitFlags	Gets the initialization flags. Internal use only.
Operation	

Terminate	<p>Terminates the engine and closes all active sessions and database connections. Must be called before the application is closed.</p> <p>Note: Unlike other objects, SmEngine is released from memory only when you call the SmEngine.Terminate method.</p> <p>Note: If the SmEngine.Terminate method is not called before the application is closed, the LUM license for the application will not be released.</p>
LoadLibrary	Loads a DLL into memory, and returns the module handle for the DLL. Using this function eliminates problems related to short vs. long DLL filenames, and to incorrect handling of floating point operations by some compilers.
CreateObject	Creates and returns a reference to an Automation object

Obtaining the ISmEngine Object

To create and initialize an ISmEngine Object:

```
Dim Engine as SmApplic.SmFreeThreadedEngine
    ' create SmarTeam engine object

Set Engine = CreateObject("SmApplic.SmFreeThreadedEngine")
    ' initialize object

SmEngine.Init "SmTeam32"
```

Adding an Object using SmartBasicScript Editor

When using a script in SmartBasicScript Editor which needs to create another SmSession, it is recommended to use the following procedure:

1. In the script attached to Smarteam hook instead of creating the engine always take it via the session:

```
Set SmEngine = SmSession.Engine
```
2. After creating another session, initialize it:

```
SmSession.Init SmEngine, "test", "SmTeam32"
```

Notes:

- If it is necessary to open another SmSession you must use the same (one) running SmEngine object:

```
Set SmEngine = SmSession.Engine
Set NewSmSession= SmEngine.CreateSession(...)
NewSmSession.Init SmEngine, "test", "SmTeam32"
```

- If the new session (application) is running outside the SmarTeam process (as an external application), it should be compiled using other development tools, for example, Visual Basic. The bundled SmartBasic Editor is dedicated only for in-process SmarTeam script compilation and cannot be used for external or out-of-process applications.

SmSession Object

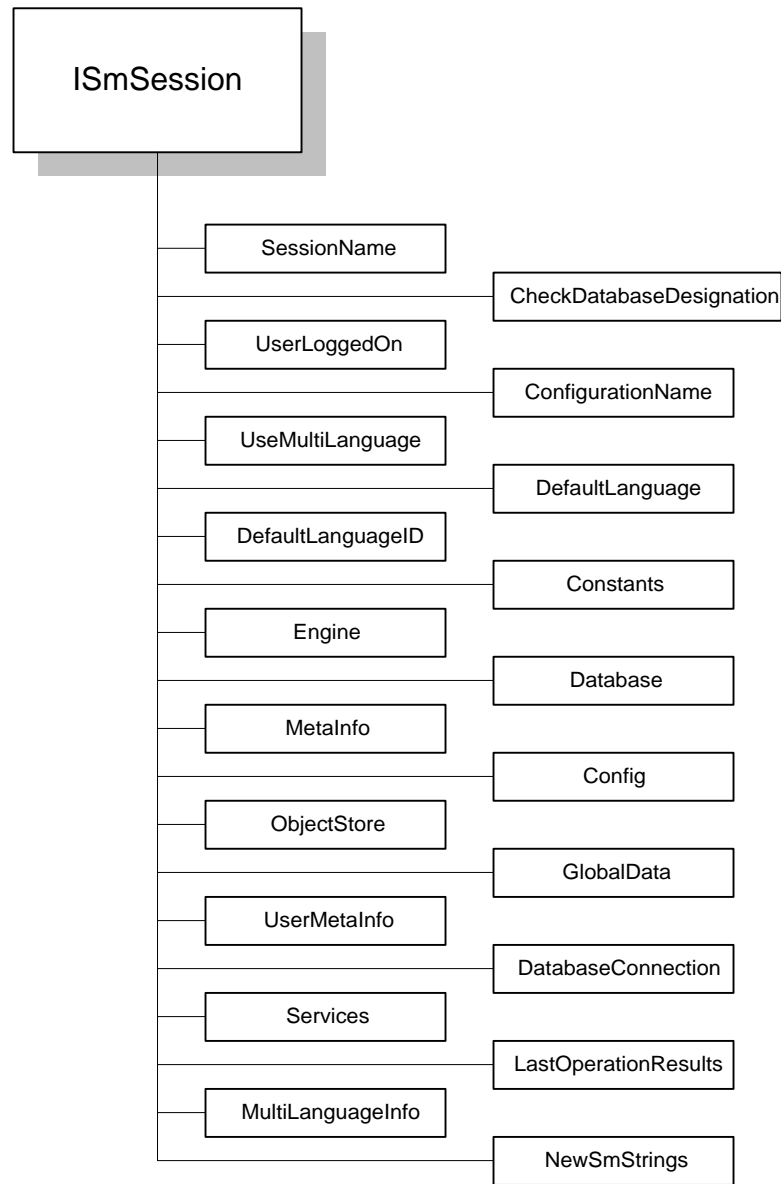
The **SmSession** object represents a session within the **SmarTeam** Engine. It is usually associated with a single user, using a single database connection.

The **SmSession** object provides the following functionality:

- User login and access to information about the user
- Access to the Database and the Database Connection
- Access to the Data Model Meta-information
- Access to the Persistent Objects ObjectStore
- Access to a collection of installable Add-in Services
- Access to a session-wide shared area for exchange of information.
- Access to the configuration parameters.

SmEngine keeps a list of all concurrently open sessions, which can be accessed using the **Sessions** property.

The SmSession and its major components is shown in the following object diagram.

*Figure 5-2 SmSession Object Diagram*

Properties

The ISmSession object has the following properties

Property	Description
Configuration	
Config	Returns an SmConfig object representing the session configuration, as ISmConfig
ConfigurationName	Returns or sets the name of the session's configuration file, for example, SmTeam32.INI. The default location for this configuration file is \SmarTeam\LocalConfig\[Database LocalName]\[User Name]/ SmarTeam32.INI
UseMultiLanguage	Set to True to enable multilanguage usage.
SessionName	Returns the universally unique name of the session.
DefaultLanguage	Returns or sets the default language for the display, for example Italiano.
DefaultLanguageID	Returns default language ID.
Database	
Database	Returns an SmDataBase object representing the connected database for the session, as ISmDatabase
DatabaseConnection	Returns an SmDataBase object that represents the database connection for the session, as ISmDatabaseConnection
CheckDatabase Designation	If set to True, an error message is issued when trying to connect to a non-registered SmarTeam database, which is not a foreign database (does not have the type sdtForeignDatabase). If set to False, no error message is issued. Default is True.
User	
UserLoggedIn	True if a user has logged on the system in this SmSession. Only one user can log on to a session.
UserMetaInfo	Returns an SmMetaInfo object representing the current user in the system, as ISmUserMetaInfo
Data Access	
GlobalData	Provides access to a session-wide shared storage area, which can be used to store and retrieve data throughout the life of the session. Data is stored as a VariantList object.
Constants	Accesses SmarTeam Enum constants from the SmApplic Library by constant name
Object Access	
Engine	Returns the parent SmEngine object, as ISmEngine
MetaInfo	Returns an SmMetaInfo object representing SmarTeam data model functionality, as ISmMetaInfo

ObjectStore	Returns an SmObjectStore object representing SmarTeam persistent object management, as ISmObjectStore
Services	Returns an SmServices collection object representing a list of in services available to the session, as ISmServices.
LastOperationResults	Returns an SmOperationResults collection object representing results of a life-cycle task operation on a set of objects, as ISmOperationResults. An item of the collection is an object together with the results of the operation on that object.
MultiLanguageInfo	Provides access to information for creating controls in different languages, as ISmMultiLanguageInfo

Methods

The ISmSession object has the following methods:

Method	Description
Configuration	
Init	Initializes the session parameters according to the session file.
Database	
OpenDatabase Connection	Opens a connection to a specified SmarTeam database, as ISmDatabaseConnection. Any previous connection is automatically released.
OpenWizardDatabase Connection	Opens a connection to a registeredWizSrc database, as ISmDatabaseConnection.
OpenForeignDatabase Connection	Opens a connection to a specified non- SmarTeam (foreign) database, as ISmDatabaseConnection.
User	
UserLogin	Logs a user into the system by user name and password.
UserLogoff	Logs the user off the session.
Object Access	
NewVariantList	Creates a new SmVariantList object, as ISmVariantList.
NewSmStrings	Creates a new SmStrings object, as ISmStrings.
GetService	Returns a specified Add-In service according to a specified ProgId.
IsServiceEnabled	Returns True if the specified Service object can be used at the present time. A Service object is considered as disabled if it is not installed, or if there is a condition that prevents it from working, such as a missing license.
Operation	
Close	Terminates the current session.

Obtaining an ISmSession Object

To create and initialize an ISmEngine Object:

```
Dim Engine as SmApplic.SmFreeThreadedEngine

Dim Session As SmApplic.SmSession

' create SmarTeam engine object

Set Engine = CreateObject("SmApplic.SmFreeThreadedEngine")

' initialize object

SmEngine.Init "SmTeam32"
```

```
Set Session = CreateObject("SmApplic.SmSession")  
  
Session.Init(Engine, "MySession", "Snteam32")
```

SmDatabase Object

The **SmDatabase** object contains information about a specific database, such as the database name, language and version, and information about active connections.

The **SmDatabaseConnection** object represents a connection to a database, and provides database-related functionality such as the ability to insert, update and delete tables, and query functions.

Connection to the database is created when you call the method **SmSession.OpenDatabaseConnection**.

The SmDatabase and its major components is shown in the following object diagram.

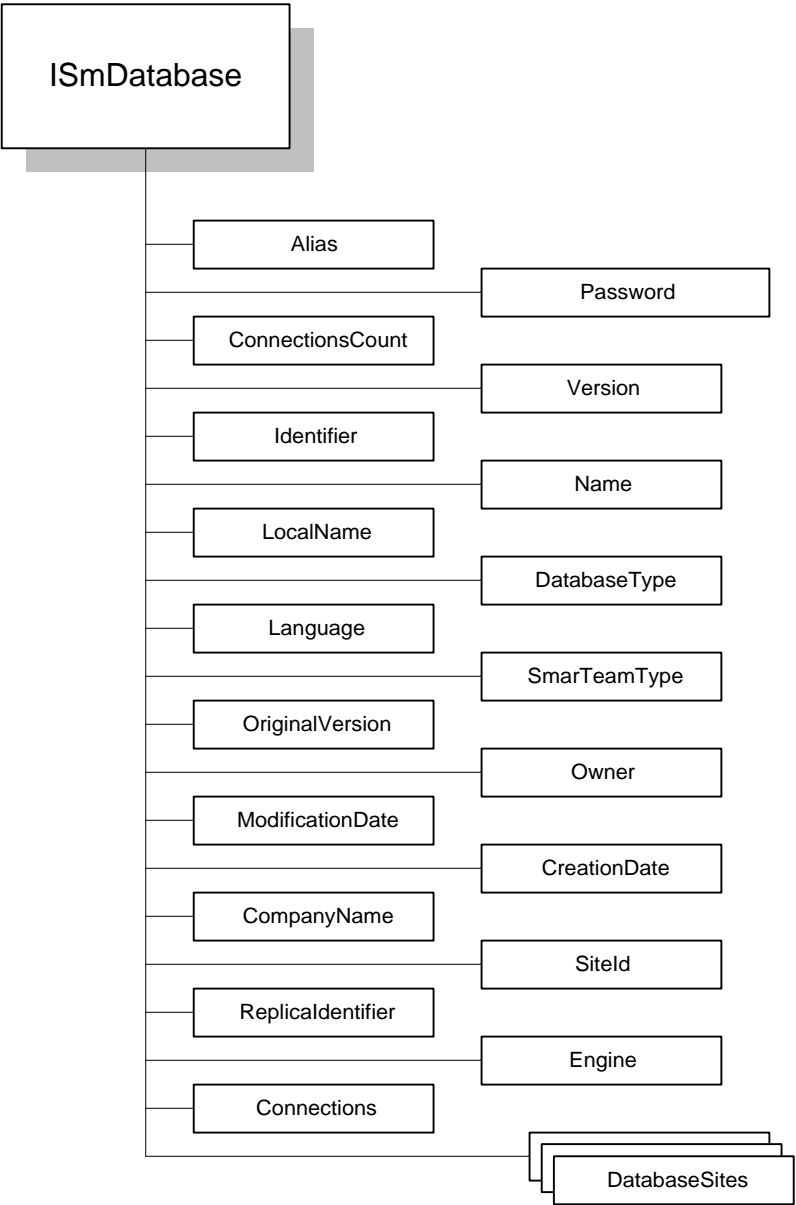


Figure 5-3 SmDatabase Object Diagram

Properties

The ISmDatabase object has the following properties:

Property	Description
Internal Database Data	
Password	Returns the internal database password.
Version	Returns the SmarTeam database version.
Identifier	Returns the internal database identifier.
Name	Returns the internal database name.
DatabaseType	Returns the database type.
Language	Returns the database language.
OriginalVersion	Returns database original version.
Owner	Returns database owner.
ModificationDate	Returns the date of the last database modification.
CreationDate	Returns the creation date of the database.
CompanyName	Returns the internal database company name.
SiteId	Returns identifier of the site for a replicated database.
ReplicaIdentifier	Returns the replica identifier.
Non-Internal Database Data	
Alias	Returns the alias or connection string of the database.
LocalName	Returns the database local name – the name of the directory for configuring SmarTeam by database.
SmarTeamType	Returns SmarTeam database type, as SmarTeamDatabaseTypeEnum.
ConnectionsCount	Returns number of open connections to the database.
Object Access	
Engine	Returns an SmEngine object representing the parent engine, as ISmEngine.
Connections	Returns an SmDataBaseConnection object representing one of the connections to the database, as ISmDatabaseConnection.
DatabaseSites	Returns a collection of all replicated database sites, as ISmDatabaseSites.

Methods

The ISmDatabase object has the following methods:

Method	Description
ReloadInfo	Reloads database information.
CloseConnections	Closes all open connections to the database.

Obtaining an ISmDatabase Object

The following sample code creates a connection to a database, and performs a login to the database:

```
Sub main()  
  
Dim Engine As SmApplic.SmEngine  
  
Dim Session As SmApplic.SmSession  
  
Dim FirstDB As SmApplic.SmDatabase  
  
Set Engine = New SmApplic.SmEngine  
  
Engine.Init "SmTeam32"  
  
Set Session = Engine.CreateSession("DemoApplication",  
Engine.ConfigurationName)  
  
' Connect to the first database  
  
Set FirstDB = Engine.Databases(0)  
  
Session.OpenDatabaseConnection FirstDB.Alias,  
FirstDB.Password, True  
  
' Login  
  
Session.UserLogin "joe", ""  
  
...  
  
Engine.Terminate  
  
End Sub
```

SmConfig Object

The **SmConfig** object allows retrieval and manipulation of SmarTeam configuration data. You can edit the configuration data, including reading, writing and deleting either individual configuration data items or entire configuration sections.

System Configuration Service

Beginning with V5R13, configuration settings are handled by the System Configuration Service. This service replaces the various SmarTeam configuration location (INI files, Registry) used in previous versions of SmarTeam. For more information on the System Configuration Service and its parameters, see “System Configuration Service” in the SmarTeam - Editor Online Help.

Using INI Files

With the System Configuration Service you should continue using configuration INI files in their same locations. However you should be aware that SmarTeam now handles INI files *according to their file names* in the following way:

If the INI file has a standard SmarTeam INI file name, such as `SmTeam32.ini`, SmarTeam gets the configuration settings from the System Configuration Service. The contents of the INI file are ignored.

If the INI file does not have a standard SmarTeam INI file name, for example, `myname.ini`, SmarTeam will take the configuration settings from the INI file and not from the System Configuration Service.

Using the API to access SmarTeam Configuration Data

With the System Configuration Service, there is no change in how you access SmarTeam configuration data. You continue to use the same key path, as described below, where now SmarTeam accesses the System Configuration Service, transparently to the user.

Note: Since the location of the configuration parameters may change between versions of SmarTeam, you should not attempt to access the Registry directly.

Accessing Non-SmarTeam Data

In cases where you specify a key path for a registry location, which does not correspond to a SmarTeam configuration item, SmarTeam will access the registry and not the System Configuration Service. Similarly, if you have provided a non-standard INI file name, as above, a key path to the INI file will access the data in the INI file.

Configuration Types

In order to provide maximum flexibility, SmarTeam can be configured in the following independent ways:

- **Admin Configuration** - determines the configuration for all the **SmarTeam** users associated with it.
- **Local Configuration** - determines the configuration for all the users using a specific workstation
- **Database-Specific Configuration** - determines the configuration for all users connected to a specific database
- **User-Specific Configuration** - determines the configuration for a specific logged-on user connected to a specific database

Accessing ISmConfig

You can create an SmConfig object as follows:

```
Dim SmConfig As SmApplic.SmConfig  
  
Set SmConfig = CreateObject("SmApplic.SmConfig")
```

You can access most of the configuration data through the SmConfig object without having to create an SmEngine or SmSession object.

For example:

```
MsgBox SmConfig.HomeDirectory & "\SmarTeam  
  
MsgBox SmConfig.IniFileName & "SmTeam32.ini  
  
MsgBox SmConfig.Value("$Local\Init Coordinates\Maximized") & "YES"
```

Other configuration data only becomes available after you have created an SmEngine object, an SmSession object, or a user has logged in. That configuration data is available only through SmSession.Config.

If you create an SmSession object, you can always access configuration data through SmSession.Config so it is unnecessary to create a separate SmConfig object in that case.

Accessing Configuration Data

Key Path

You address configuration information by using a key path (also called an option path) of the form:

```
$<Option type>\<Section name>\<Identifier name>
```

For data in a configuration ini file, the `Option` type can be `Admin`, `Local`, `Database` or `User`, depending on which type of configuration is desired.

For data in the Registry the `Option` type can be `RegClassesRoot`, `RegCurrentUser`, `RegLocalMachine`, `UserReg`, or `AdminReg`.

You can also use a parameterized key path to add flexibility in specifying the configuration item key path.

`ISmConfig` provides many properties and methods for editing configuration data; they are all based on addressing the data via a key path.

See the COM API Reference Guide for detailed information.

Examples

For example, you can access an individual configuration data value by the following command:

```
MyOption = Session.Config.Value(<Key path>)
```

1. The following command reads the value of the `<MyIdent>` identifier from the `<MySection>` section of the **SmTeam32.ini** file located in the **LocalConfig** directory:

```
MyOption = Session.Config.Value("$Local\<MySection>\<MyIdent>")
```

2. The following command tries to read the value of the `<MyIdent>` identifier from the `<MySection>` section of the **SmTeam32.ini** file located in the **User** subdirectory:

```
MyOption = Session.Config.Value("$User\<MySection>\<MyIdent>")
```

If not found, the system tries to read the same identifier from the **SmTeam32.ini** file located in the **LocalConfig** directory.

ExpandValue Property

Important Note: It is not recommended to adjust the ExpandValue Property of the ISMConfig Interface.

Remarks

The ExpandValue gives the same results as the method ISMConfig.Value, specifying the configuration item key path.

Instead of using a "hard-wired" key path, you can parameterize some or all parts of the key path. That lets you use the same key path to address a related set of configuration items. You address individual configuration items by setting the values of the key path parameters.

For example, the path:

```
$RegLocalMachine\SoftWare\Smart Solutions\SmarTeam\Database  
Connection Setup\ $Num
```

has its last section parameterized (\$Num). By assigning values to \$Num, for example, "1", "2", "3", you address different configuration items without changing the form of the key path.

Any part of the key path except for the first can be parameterized by replacing the part name by any variable name prefixed by "\$". You can parameterize more than one part of the path at a time. The parameter values are represented by a Variant array, which contains as many elements as there are parameters in the key path. In the above example, you use an array of one element.

After assigning parameter values directly into the array, you call ExpandValue with the key path and parameter array as arguments.

Common Parameterized Key Paths

Warning: Do not write to these locations unless you are sure of the results.

Description	Key
GlobalDatabaseConnectionNum	'\$AdminReg\Database Connection Setup\ \$Num
GlobalDatabaseName	'\$AdminReg\Database Connection Setup\ \$Num\Database Na

Editing Configuration Data in the Windows Registry

You use the same type of key path to address configuration data in the Registry, except that the Option type can be **AdminReg** or **UserReg**, corresponding to Admin or User configuration data:

Examples

1. The command

```
MyOption = Session.Config.Value("$AdminReg\ <MySection>\<MyIdent>")
```

reads the value of the <MyIdent> identifier from the
HKEY_LOCAL_MACHINE\SOFTWARE\Smart
Solutions\SmarTeam\<MySection> path of the Windows registry:

2. The command

```
MyOption = Session.Config.Value("$UserReg\ <MySection>\<MyIdent>")
```

tries to read the value of the <MyIdent> identifier from the
HKEY_CURRENT_USER\SOFTWARE\Smart Solutions\SmarTeam\<MySection> path
of the Windows registry:

If not found, the system tries to read the identifier from the
HKEY_LOCAL_MACHINE\SOFTWARE\Smart Solutions
\SmarTeam\<MySection> path of the Windows registry.

3. The three option types \$RegClassesRoot, \$RegCurrentUser, and
\$RegLocalMachine, enable you to read from any path of the Windows
registry.

The command

```
MyOption = Session.Config.Value("$RegClassRoot\ <MySection>\<MyIdent>")
```

reads the value of the <MyIdent> identifier from
HKEY_CLASSES_ROOT\<MySection> path of the Windows registry:

Properties and Methods for Editing Configuration Data

The following is a list of the methods and properties available for editing
configuration data in .ini files or in the Registry. See the COM API
Reference Guide for details and examples of these methods and properties.

Terminology

Configuration Item := {Identifier = Value}

Configuration Section := {[SectionName] + Configuration Items}

Properties and Methods

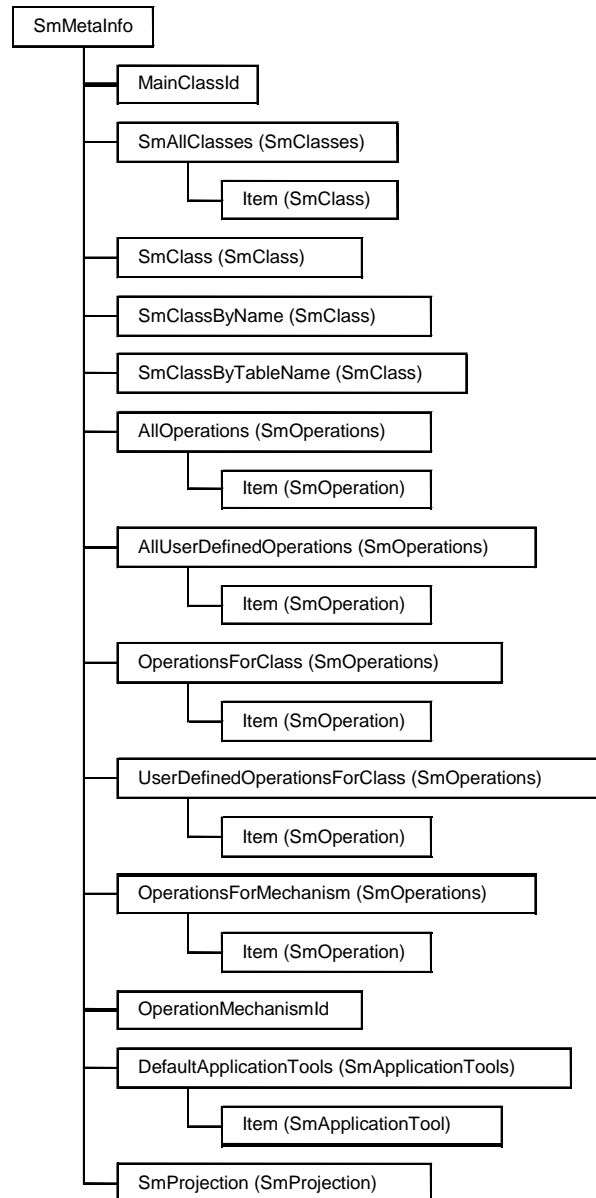
Property or Method	Description
[Expand]Value	Sets or gets a Configuration item value using [parameterized] key path
Read[Expand]ValueAs[Type]	Gets a Configuration item value as Type using [parameterized] key path. If the configuration item is found, the default value specified is returned.
ReadExisting[Expand]ValueAs[Type]	Gets a Configuration item value as Type using [parameterized] key path. If the configuration item is found, an error message is issued.
Read[Expand]Section	Gets all Configuration items of a section as SmString using [parameterized] key path
IdentsInSection	Gets all Configuration items of a section as SmRecordList using a fixed key path
Read[Expand]Registry	Gets all Configuration items of a registry section as SmStrings using a [parameterized] key path
Write[Expand]Section	Writes all Configuration items of a section as SmString using [parameterized] key path
Delete[Expand]Value	Deletes an individual configuration item using a [parameterized] key path
Delete[Expand]Key	Deletes an individual configuration item using a [parameterized] key path (Same as DeleteValue)
Delete[Expand]Section	Deletes a configuration section using a [parameterized] key path

Note: When using the function Session.Config.ReadSection, the Key is always returned in lowercase.

Metadata Management Objects

The metadata management objects of the **SmarTeam** Engine library contain information relating to the **SmarTeam** data model.

The main properties and object hierarchy of the metadata management objects are shown below:



SmMetaInfo Object

The **SmMetaInfo** object represents all information about a specific **SmarTeam** data model. Each **SmSession** object contains an **SmMetaInfo** object for the connected database. The **SmMetaInfo** object contains several properties that represent specific terms in the system, including:

- Persistent Classes represented by the **SmClasses** and **SmClass** objects.
- System and user-defined operations, defined in the database, represented by the **SmOperations** and **SmOperation** objects.
- Applications tools represented by the **SmApplicationTools** and **SmApplicationTool** objects.
- Projections, represented by the **SmProjection** object.

Important Note: In a script, `SmSession.SmMetadata` can be accessed only after calls to `SmSession.OpenDatabaseConnection` and `SmSession.UserLogin` have been made.

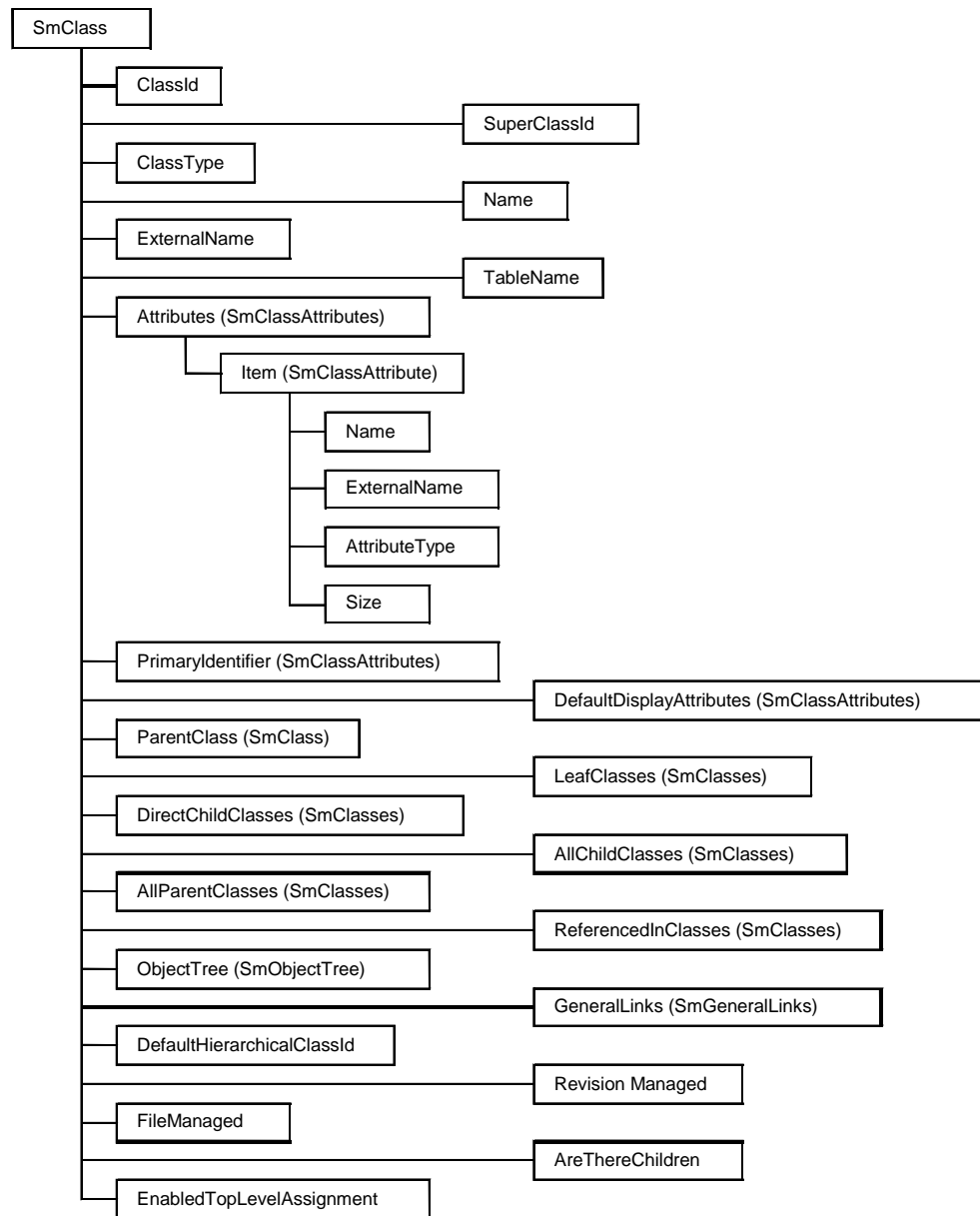
SmClasses and SmClass Objects

The **SmClass** object represents a Persistent Class defined in the **SmarTeam** Data Model. **SmClasses** is a collection object representing a set of **SmClass** objects.

The **SmClass** object provides the following objects and corresponding functionality:

- Attributes in the class, represented by the **SmClassAttributes** and **SmClassAttribute** objects
- Object tree, represented by the **SmObjectTree** object. This object defines the Persistent Classes which can be linked as parents or children to instances of the class.
- General links, represented by the **SmGeneralLinks** object. This object defines the Persistent Classes which can be linked to instances of the class.

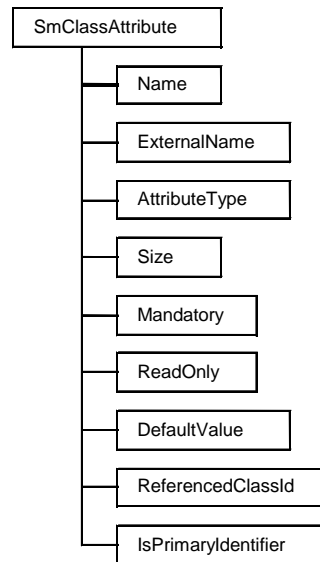
The main properties and object hierarchy of the **SmClass** object are shown below:



SmClassAttributes and SmClassAttribute Objects

The **SmClassAttribute** object represents an attribute of a Persistent Class defined in the **SmarTeam** Data Model. **SmClassAttributes** is a collection object representing a set of **SmClassAttribute** objects.

The main properties and object hierarchy of the **SmClassAttribute** object are shown below:



Reference to Class

A Class Attribute is called “Reference to Class” when it represents a reference from the current class to another class (the “referenced class”). The `ClassId` of the referenced class is represented by the `ReferencedClassId` property of the Reference to Class. The purpose of this ability is to allow an object of a class to reference another object through one of its properties, usually for displaying properties of the referenced class.

For example, the Class Attribute “`USER_OBJECT_ID`” of a Document or Folder is a Reference to Class, which represents a reference to the User Class; its `ReferencedClassId` property has the value of the `ClassId` of the User Class. A Folder object would use this attribute to refer to the User object that is its creator.

Example

In the following example, `FolderObject` is an `SmObject` of class `Folder`. You obtain the Class Id of the creator User class as follows:

```
UserClassId =  
FolderObject.SmClass.Attributes.ItemByName( "USER_OBJECT_ID" ).ReferencedClassId
```

The `AttributeType` (type of the value that is stored in this attribute) of a Reference to Class is `sdtObjectIdentifier`. The Reference to Class attribute of an `SmObject` contains the Object Id of the referenced `SmObject`.

The `ISmClass.ReferencedInClasses` Property and the `SmObject.CheckReferences` method check if a class or object is referenced by another.

SmProjection Object

A Projection object is a subset of the Class Attributes of an SmClass and is used to represent an SmObject by these attributes. One main use of Projection objects is to display the attributes of a referenced class. For example, when you display the properties of a Folder object on a Profile Card and you want to include the LOGIN property of the User who created the Folder (where the User class is referenced by the Folder, as described in the previous section), you would use a Projection. The Projection object is created for the User Class and selects only the LOGIN property of the User Class.

Example

The following is an example of how to create a Projection for the UserClass with the ClassId “UserClassId” mentioned above, give it a name “UserProjection”, and save it in the Database:

```
Set SmAttributes = SmSession.MetaInfo.NewClassAttributes(UserClassId)

SmAttributes.Add "LOGIN"

SmAttributes.Add "USER_TITLE"

' create projection for the user class

Set UserProjection = SmSession.MetaInfo.NewProjection(UserClassId)

UserProjection.Fields = SmAttributes

UserProjection.Separator = ", "

UserProjection.Name = "UserProjection"

UserProjection.Insert
```

Any time you want to use this projection you refer to it by the name UserProjection, similar to the names Folder and Document.

Note that you can define more than one Projection for a Class, which allows different kinds of property displays for a Class depending on your requirements.

SmClassReferenceObject Object

The SmClassReferenceObject object is used to display the properties of a specific object of a Class for a specific Projection of the Class.

Example

To display the "UserProjection" Projection properties for the User object referred to by the Folder object FolderObject, you use the function GetSmClassReferenceObject as follows:

```
' Get the Projection Id for the UserProjection object

UserProjectionId = MetaInfo.SmProjectionByName(UserClassId,
"UserProjection").ID
```

```
` Get the Object Id of the User object, from the Reference to Class attribute  
of ` the Folder object
```

```
UserObjectId = FolderObject.Data.ValueAsInteger("USER_OBJECT_ID")
```

```
` Use the GetDisplayValue method to display the Projection values
```

```
ProjectionString =  
SmSession.ObjectStore.GetSmClassReferenceObject(UserClassId, UserObjectId,  
UserProjectionId).GetDisplayValue(True)
```

```
`Results: 'joe, Chairman of the Board
```

For each Class, a default Projection is defined. If you set UserProjectionId to 0 in the code above, the default projection values will be displayed.

To display the Projection attributes when you already have the User object, you use the function SmClassReferenceObject as follows:

```
MsgBox  
UserObject.SmClassReferenceObject(UserProjectionId).GetDisplayValue(True) `   
joe, Chairman of the Board
```

Reference to SmLookup Class

The Reference to Class ClassAttribute is frequently used with a Lookup class as the referenced class.

A Lookup Class is a collection of predefined objects any of which may need to be accessed by another class. One example of a Lookup class is File Type, which contains objects corresponding to each of the possible file types that can occur in an application, such as SolidWorks Part and Microsoft Word.

Example

This example uses the Reference to Class attribute FILE_TYPE to access the file type of a Document object from the Lookup class File Type.

Assuming you have an SmObject of the Document class:

```
` Get the Lookup object Id - the default is Microsoft Word
```

```
Lookup_Object_Id = SmObject.Data.ValueAsString("FILE_TYPE")
```

```
` Get the referenced Lookup Class Id
```

```
LookUpClassId =  
SmObject.SmClass.Attributes.ItemByName( "FILE_TYPE" ).ReferencedClassId  
  
' Get the LookUp object's display  
  
LookUpValueStr = SmSession.ObjectStore.GetSmLookUp(LookUpClassId,  
LookUp_Object_Id).DisplayName
```

SmObjectTree Object

The **SmObjectTree** object represents composition rules for a specific Persistent Class in the **SmarTeam** data model. The **SmObjectTree** object is referenced by an **ObjectTree** property of a **SmClass** object.

Example

The following example prints all possible child instances for a **SolidWorks Assembly** class instance:

```
Sub Test()  
  
    Set SWAssemblyClass = SmSession.MetaInfo.SmClassByName("SolidWorks  
Assembly")  
  
    Set CompClasses = SWAssemblyClass.ObjectTree.GetChildClasses  
(SWAssemblyClass.DefaultHierarchicalClassId)  
  
    If Not (CompClasses Is Nothing) Then  
  
        count = CompClasses.count - 1  
  
        For i = 0 To count  
  
            Set SingleClass = CompClasses.Item(i)  
  
            MsgBox "Possible child: " +_ SingleClass.ExternalName  
  
        Next  
  
    End If  
  
End sub
```

Class Behaviors

Class behavior refers to a common functionality that can be imposed on a class.

Examples of such functionality are:

- The ability of an object of the class to be associated with a file (file-managed)
- The ability of an object of the class to have successive revisions (revision-managed)
- Restrictions on possible links to objects of the class

SmarTeam provides two different mechanisms to impose a class behavior on a class:

- Class-level behavior
- Optional class-level behavior

Class-Level Behavior

A class-level behavior (CLB) specifies a special common functionality for a class. In addition, it optionally specifies a set of class attributes the class must possess in order to be able to support the required common functionality. For example, the File Control CLB specifies the functionality and attributes required for a class to be associated with a file. It specifies a mechanism for the class to be aware of the file, to point to the physical file on disk or vault, and to prompt to delete the file when object is deleted. The class must have a File Name attribute and a Directory attribute. A CLB imposed on a class can also restrict certain methods from acting on an object of the class.

The CLBs are defined separately in the data model and are standardized across classes. One or more CLBs can be associated with a class, and the same CLB can be associated with two or more classes. Two basic CLBs are File Control and Revision Control; a class that has these CLBs is called file-managed and revision-managed, respectively. The Folder and the Document classes in the SmDemo database are both file-managed and revision-managed.

Once a CLB is associated with a class in the data model, all objects of that class must conform to the CLB behavior.

Note: A CLB can be defined in the data model for a link class as well as for a regular class. This allows you to assign specific behaviors to links at the class level. One example of a CLB applied to a link class is the Link Direction CLB, which includes the directionality of the link.

Note: In previous versions of SmarTeam, Class-Level Behavior was referred to as a Class Mechanism.

States and CLBs

A set of states can be associated with a CLB. When such a CLB is specified for a class, all objects of the class can be in one of the states of the set.

The following rules concern states and CLBs:

- The states of the CLB override any “native” states that might have been assigned to the class, for example, New, CheckedIn, CheckedOut, etc. for the Document class.
- If no states are associated with the CLBs imposed on a class, the “native” states of the class will obtain.
- Two CLBs, each associated with a different set of states, cannot be assigned to the same class
- API methods are available for accessing the states associated with a CLB or class, see [API Methods](#) below.

Optional Class-Level Behavior

SmarTeam supports a second type of class behavior, the Optional Class-Level Behavior (OCB). As opposed to a CLB, to which all objects of the class must conform, an OCB is *optionally* imposed on an object of the class. The decision whether to impose an OCB on a particular object of a class is made when the object is created at run time. If it is decided to use the OCB, it is imposed then on the object, in addition to any CLBs that may have been imposed on the class. Accordingly, different objects of the same class can have different OCBs imposed on them.

One or more of the set of class OCBs can be assigned directly to a persistent object of the class using the Add method of the ISmSupportedClassMechanisms interface, accessible through the OptionalClassMechanisms property of the SmObject that represents the persistent object.

The collection of all Class Behaviors, including both CLBs and OCBs, that are imposed on a specific object are called *Supported Class Behaviors*. The collection object `ISmSupportedClassMechanisms` is used to hold the collection. The object is said to *support* the class behaviors.

Note: In V5R11, only one OCB can be assigned to an object.

Benefit of Optional Class-Level Behaviors

The major benefit of the OCB is that, by assigning them different OCBs, two objects of the same class can have different class behaviors. This allows you to define a wide class that represents a generic object, such as a Part CAD Component class, which is independent of any particular CAD product, and then to distinguish, on the object level, between different types of CAD Documents, such a SolidWorks Part and Solid Edge Part.

The OCB imposed on an object can influence it in the following ways:

- Determines the possible link classes that can link to and from the object
- Determines the object's life cycle operations characteristics (life cycle rules)
- Determines which of the generic class attributes are relevant to the particular object

In general, an object can have more than one OCB, where each OCB can specify a type of behavior, for example, one OCB for possible links and one OCB for relevant class attributes.

Example

In this example, two Document objects are created, each with a different OCB, similar to the previous figure.

```
Dim ClassId As SmallInt

Dim SWPart, SEPart As ISmObject

Dim SW_PART_Mechanism, SE_PART_Mechanism As ISmClassMechanism

'Create Document object

SWPart = SmSession.ObjectStore.NewObject(ClassId)

'Get SolidWorks Document behavior
```

```
SW_PART_Mechanism = MetaInfo.ClassMechanismByName("TDM_SW_PART")  
  
'Add it to Document to make it a SolidWorks Document  
  
SWPart.OptionalClassMechanisms.Add(SW_PART_Mechanism)  
  
'Create Document object  
  
SEPart = SmSession.ObjectStore.NewObject(ClassId)  
  
'Get Solid Edge Document behavior  
  
SE_PART_Mechanism = MetaInfo.ClassMechanismByName("TDM_SE_PART")  
  
'Add it to Document to make it a Solid Edge Document  
  
SWPart.OptionalClassMechanisms.Add(SE_PART_Mechanism)
```

Link Composition

Link Composition is the determination of an appropriate and permissible link class to link two objects.

Two common composition activities are:

- Determining the permissible link classes to link an object to an undetermined second object.
- Determining the permissible link classes to link two objects that belong to different classes.

Link Classes

To choose an appropriate link class for a link composition, three things should be taken into account:

- The availability of the link class in the data model
- The permissibility of the link class for the linked objects.
- The characteristics of the desired link class

Availability of the link class in the data model

You can only use link classes that have been predefined in the data model. See below for a discussion of how link classes are defined in the data model and how you can use API functions to get the possible links.

Permissibility of link class for the linked objects

You can only use link classes that are permissible for the objects to be linked. Permissibility requirements can exist both on the class and the object level, as discussed below.

Link Class Characteristics

Using API functions, you can specify desired characteristics of the required link class:

Direction of the link – The directionality of a link class is specified by the assignment of the Directional Link CLB to the class. For example, a hierarchic link class such as Parent-Child or Child-Parent has the Directional Link CLB assigned to it. The actual direction of the link in a specific link object whose class has the Directional Link CLB is determined by the order of the linked object arguments in the function that creates the link object. For example, in the function `SmObject1.LinkOneLevel(LinkClassId, SmObject2, LinkAttributes)`, the link direction is from `SmObject1` to `SmObject2`.

Regular attributes of the link class – you can specify additional attributes of a link class by specifying the regular attributes of the link class, which are defined in the data model.

Link Classes in the Data Model

This section describes how link classes are defined in the Data Model. This information is necessary to understand how the API supports compositions.

Link class definitions by Data Model Designer

The Data Model Designer automatically establishes a link class between the main (Project) class and every other superclass. These link classes do not have the LinkDirection CLB imposed and hence are non-directional.

The Data Model Designer can be used to establish a “user-defined” link class between any two superclasses, including defining a link class between a superclass and itself. Only one link object of a link class can be created for a pair of objects of the two superclasses, unless an index attribute is added to the link class, as described in the next paragraph. This “user-defined” link class can optionally have the LinkDirection CLB imposed. Only one “user-defined” link class can be defined for each pair of superclasses.

For the user-defined link class of the previous paragraph, you can create more than one link object between a specific pair of objects if you define an index attribute in the user-defined link class. The additional link objects are differentiated by the index attribute value.

One hierarchic link class is automatically established for each superclass, to link objects within that superclass. It can be used for linking two objects of that superclass only. For example: assembly and part in the Document superclass. This link class has the LinkDirection CLB imposed.

Link class definitions introduced by Integration Modules

The integration modules (the integration module is used as an example, the same is true for other modules) can add link classes to the Data Model, in addition to the user-defined link class described in the previous section. Integration link classes are link classes within one superclass. The user cannot modify Integration link classes.

These integration link classes are set up to be used in CLB compositions, as described below. A CLB Relations table associates a pair of OCBs with each integration link class, meaning that the link class can link only objects with the specified OCBs.

The integration modules do not add hierarchic links. The integrations must use hierarchic link classes that were defined for the superclass by the Data Model Designer. It is apparently not a restriction to use the same hierarchic link class to link different pairs of OCBs.

Permissible Compositions

As mentioned above, you can only use a link class for a composition if the link class is permissible for the objects to be linked.

Two levels of permission are required:

- **Class-Level Permission** – The link class must be permissible for the objects’ superclasses, that is, it must be defined in the Data Model as a link class that links the superclasses. In the case of hierarchical links, the link class must be permissible for the objects’ concrete leaf classes
- **Object-Level Permission** – The link class must be permissible for the OCBs that are imposed on the objects to be linked, that is, the link class must be associated with the objects’ OCBs in the CLB Relations table that associates pairs of OCBs with link classes.

Getting Permissible Link Classes for Compositions

API methods are provided to get the permissible link classes for linking given objects.

A new group of methods is provided in ISmObjectStore to get possible Link Classes that can link specific objects. These methods take into account OCBs that may have been assigned to the objects (see [API Methods](#) below).

To get a hierarchic link class to link objects of a specified parent class and a specified child class, use the HierarchicalLinkClassByClasses method to get a hierarchic link class that exists between objects of the parent class and child class.

Class Composition

Class Composition is a special case of the Link Composition discussed in the previous sections. In Link Composition, the link class must have both class-level and object-level permissions, as discussed in the section “Permissible Compositions”. In a Class Composition, only class-level permission is required for a link class; object-level permissions are ignored. The permissible link classes of a Class Composition depend only on the two component classes and are independent of any OCBs that may have been imposed on the two objects to be linked. The link class is determined from the two linked objects’ classes and, as a result, the link class can be used to link any two objects of the two classes. You would use Class Composition when no OCBs have been assigned to the objects or when you wish to ignore any OCBs that have been assigned to the objects.

Getting Permissible Link Classes for Class Compositions

API methods are provided to get the permissible link classes for Class Composition.

The `SmGeneralLinks` object represents the permissible non-directional link classes that can exist between any two objects of specified classes. The `HierarchicalLinkClassByClasses` method represents the hierarchic link class that can exist between objects of a specified parent class and a specified child class, both in the same superclass. The property `SmClass.DefaultHierarchicalClassId` gets the Id of the hierarchical link class for that class.

The methods return links that do not take into account OCBs assigned to an object.

API Methods

Similar API methods are provided to handle the Link Compositions and the Class Compositions.

The following table compares the two sets of methods:

<u>Link Composition Methods</u>	<u>Class Composition Methods</u>
Requires class-level and object-level permissions	Requires class-level permissions and ignores object-level permissions
Get permissible link classes between two specified objects/classes	
ObjectStore.GetPossibleLinkClassesBetweenObjects(SmObject1 As ISmObject, SmObject2 As ISmObject, LinkDirection As LinkQueryDirectionEnum) As ISmClasses	Class1.GeneralLinks.GetLinkClasses (Class2Id) As ISmClasses SmMetaInfo.HierarchicalLinkClassByClasses (Class1Id, Class2Id) As Integer
When one object/class and the link class is specified, get the permissible classes for the related object/class	
ObjectStore.GetPossibleLinkedClasses(SmObject As ISmObject, LinkClassId As Integer, LinkDirection As LinkQueryDirectionEnum) As ISmClasses	Class1.GeneralLinks.GetRelatedClasses(LinkClassId As Integer) As ISmClasses.
Get all permissible link classes to/from one object/class	
ObjectStore.GetPossibleLinkClassesForObject(SmObject As ISmObject, LinkDirection As LinkQueryDirectionEnum) As ISmClasses	Class1.GeneralLinks.Classes As ISmClasses
Get possible OCBs for related object	Get CLBs imposed on class
ObjectStore.GetPossibleLinkedClassMechanisms(SmObject1 As ISmObject, LinkClassId As Integer, LinkDirection As LinkQueryDirectionEnum) As ISmClassMechanisms	Class1.FileManaged Class1.RevisionManaged Class1.MechanismManaged(MechanismId)

States and CLBs

The following methods are provided to get the states associated with a CLB and a class:

In SmApplic.IsmMetaInfo:

Method	Description
StatesForMechanism	Returns the set of state objects supported by the specified mechanism (CLB). Returns ISmObjects
StateMechanismId	Returns the Id of the mechanism (CLB) that supports the State specified by its id. Returns Integer

StatesForClass	Returns the set of state objects supported by the specified class The set of states depend on the CLB of the class. Returns ISmObjects
GetDefaultStateForClass	Returns the initial state object for the specified class. Returns ISmObject

Examples

The following example uses the `GeneralLinks` interface to print the default link class between classes `Project` and `Documents`:

```
Sub Test()  
  
Set DocumentClass = SmSession.MetaInfo.SmClassByName("Documents")  
  
Set ProjectClass = SmSession.MetaInfo.SmClassByName("Projects")  
  
Set LinkClass =  
  
DocumentClass.GeneralLinks.GetLinkClasses(ProjectClass.ClassId).Item(0)  
  
MsgBox "General link:" + LinkClass.ExternalName  
  
End Sub
```


This example demonstrates how to retrieve the linked objects (with reverse link) to the current object.

```
`CurrentObject is received from outside with mechanism already set

LinkClassesReverse =
SmSession.ObjectStore.GetPossibleLinkClassesForObject(CurrentObject,LinkQueryD
irectionEnum.lqdSecondToFirst)

For i = 0 to LinkClassesReverse.Count - 1

    linkClass = LinkClassesReverse.item(i)

    If not linkClass.isService Then

        LinkClassId = linkClass.ClassId

        SuperClassId = 5

        Roles.clear()

        QueryDefinition.clear()

        Roles.Add(SuperClassId, "S") 'add the linked super class id

        Roles.Add(LinkClassId, "L")

        'Enter the direction

        QueryDefinition.LinkQueryDirection =
LinkQueryDirectionEnum.lqdSecondToFirst

        LinkedObjects = CurrentObject.RetrieveRelationsAndLinks(QueryDefinition)

    End If

Next
```

Persistent Object Management

The Persistent Object Management support in the **SmarTeam** Engine library enables the creation, update and deletion of Persistent Objects, and the retrieval of information about these objects.

SmObjectStore Object

The **SmObjectStore** object contains the functionality used to manage Persistent Objects. It is used to:

- Retrieve information from the database about Persistent Objects
- Create and update Persistent Objects
- Create query objects.

SmObject and SmObjects Objects

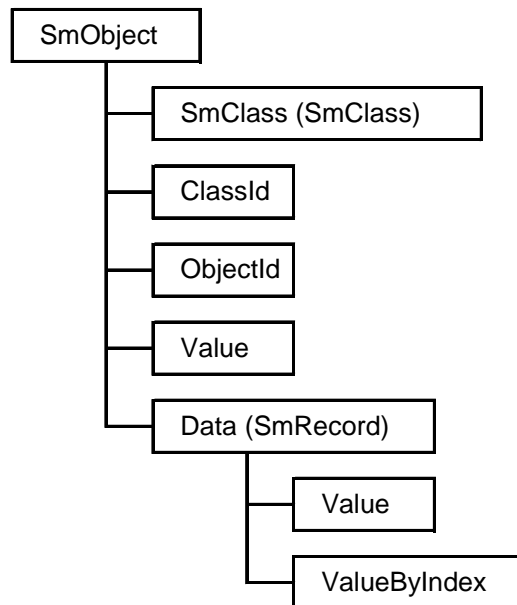
SmObject represents a single Persistent Object in **SmarTeam**. The object provides the following functionality:

- Data retrieval for the object
- Manipulation of the object, for example, **Insert**, **Update**, **Delete**
- Retrieval of other objects related to the object.

SmObject provides an in-memory representation of a Persistent Object. The Persistent Object may match an existing object that is stored in the database, or it may be a new object that has not yet been stored in the database.

SmObject objects that match existing objects in the database are returned using the various retrieval methods provided by **ObjectStore** and by other objects. Several other methods, such as the **SmObjectStore.NewObject** method, create **SmObject** objects that might not have counterparts in the database.

The main properties and object hierarchy of the **SmObject** object are shown below:



The main properties of **SmObject** are:

- **ClassId**: represents the object's class ID
- **ObjectId**: represents the object's ID
- **SmClass**: a reference to the **SmClass** object that represents the class of the object
- **Value**: used for retrieving and settings the attributes of the object. Provides a shortcut to the underlying **SmRecord** object, which is accessed through the **Data** property
- **Data**: a **SmRecord** object which contains an in-memory copy of the object's attributes. The attributes can be retrieved and set using the usual methods of **SmRecord**.

Changes to the attributes of a **SmObject** object do not affect the data stored in the database for this object. To commit the changes to the database, call one of the relevant methods, such as **Insert**, **InsertEx**, **Update**, **UpdateEx** and others.

The **SmObject** object stores the information in memory in a **SmRecord** object. The **Data** property provides access to this **SmRecord**, and the **Values** property provides a shortcut for retrieving and setting the values of the object's properties.

Creating a New Persistent Object via the SmarTeam Object Model

In order to create a new Persistent Object, follow these steps:

Create an in-memory representation of the object using the **SmObjectStore.NewObject** method:

```
Set NewObject = SmSession.ObjectStore.NewObject(ProjectClassId)
```

Add the definition of the object's attributes by calling the **AddAllAttributes** method:

```
NewObject.AddAllAttributes
```

The attributes are added to the object with an empty (null) value.

If you want to set only some of the object's attributes, you can use one of the following methods:

- The **AddAttributes** method to add individual attributes. To add more than one attribute, separate the attribute names with a semicolon ";".
- The **AddPrimaryAttributes** to add only the primary attributes of the object.
- The **AddProjectionAttributes** to add the attributes associated with a specific projection, and so on.

Set the values of the object's attributes:

```
NewObject.Value("CN_PROJECT_ID") = "Project-Test1"
```

```
NewObject.Value("CN_DESCRIPTION") = "Motor Engineering"
```

If you wish, you can first set the attributes of the object to their default values by calling the **SetDefaultValues** method.

Commit the object to the Database using the **Insert** method:

```
NewObject.Insert
```

Retrieving an Existing Persistent Object

There are several ways to retrieve a **SmObject** object from the database. The most common are:

- **SmObjectStore.RetrieveObject**: Returns the **SmObject** representing a specific Persistent Object, given a Class ID and Object ID, and retrieves the values of all the attributes.
- **SmObjectStore.RetrieveObjectByPrimaryIdentity**: Returns an **SmObject** representing a specific Persistent Object, given a Class ID and Primary ID as an **SmRecord** object, and retrieves the values of all the attributes.
- **SmObject.Retrieve**: Retrieves all the attributes of an object from the database. In order for the retrieval to be successful, **SmObject** must contain either the values of the Class ID and Object ID of the Persistent Object, or the values of the Class ID and Primary Identifier of the Persistent Object.
- **SmObject.RetrieveAttributes**: Retrieves the values only of the attributes that were added to the **SmObject**. Use this method if you need to select only specific attributes of the object, and not all attributes. Using this method improves system performance.
- Performing a query and accessing the **SmObject** instances in the query result.

Example

The following example retrieves an **SmObject** that represents an object of the class **Folder**, identified by the Primary Identity CN_ID = "Fold-0001" and Revision = "":

```
Sub Test()  
  
    Dim PrmId As SmRecList.SmRecord  
  
    Dim FolderClass As SmApplic.ISmClass  
  
    Dim FolderObj As SmApplic.ISmObject  
  
    Set PrmId = New SmRecList.SmRecord  
  
    PrmId.AddHeader "CN_ID", 40, sdtChar  
  
    PrmId.AddHeader "REVISION", 40, sdtChar  
  
    PrmId.Value("CN_ID") = "Fold-0001"  
  
    Set FolderClass = SmSession.MetaInfo.SmClassByName("Folder")  
  
    Set FolderObj = SmSession.ObjectStore.RetrieveObjectByPrimary  
    Identity(FolderClass.ClassId, PrmId)  
  
End Sub
```

Example

The following example retrieves a specific object's description using the primary identifier of the object:

```
Sub Test()  
  
    Dim SmObject as SmApplic.ISmObject  
  
    Dim SmClass as SmApplic.IsmClass  
  
    Set SmClass = SmSession.MetaInfo.SmClassByName("Projects")  
  
    Set SmObject = SmSession.ObjectStore.NewObject(SmClass.ClassId)  
  
    'Add attributes CN_PROJECT_ID, which is the primary identifier of the  
    Projects class, and CN_DESCRIPTION to the object  
  
    SmObject.AddAttributes "CN_PROJECT_ID;CN_DESCRIPTION"  
  
    ' Set value for CN_PROJECT_ID  
  
    SmObject.Value("CN_PROJECT_ID") = "Project-0003"  
  
    ' RetrieveAttributes method will search the object by it's primary  
    identifier and select value of the CN_DESCRIPTION attribute  
  
    SmObject.RetrieveAttributes  
  
    MsgBox "The description of the object is "+  
    SmObject.Value("CN_DESCRIPTION")  
  
End Sub
```

Creating an SmObject

The following can be used in various situations to create a new **SmObject** object in memory:

- **SmObjectStore.NewObject** creates a new SmObject in memory:

```
Set SmObject = ObjectStore.NewObject(ClassId)
```

- **SmObjectStore.ObjectFromData** creates an SmObject based on the input SmRecord:

```
Set SmObject = ObjectStore.ObjectFromData(Record, Disconnect)
```

- **SmObjectClone** - clones an **SmObject** from an existing object:

```
Set SmObject = SmObject1.Clone
```

Connected and Disconnected Objects

The methods used for object creation define whether the internal structure of the object can be changed, meaning whether you can add attributes to the object or delete attributes from the object.

SmObject can be created as an independent or disconnected object, or as a projection inside another **SmarTeam** object. Disconnected objects can be restructured.

Disconnected objects can be obtained by the following methods:

- **SmObjectStore.NewObject** creates a new **SmObject** not connected to any other data container.
- **SmObject.Clone** creates a copy with its own data container.
- **SmObjectStore.ObjectFromData** sets the second argument **Disconnect** to true, in order to detach the new **SmObject** from **SmRecord** container:

```
Sub Test()

    Dim ObjRec as SmRecList.SmRecord

    Dim NewObject as SmApplic.ISmObject

    Set NewObject = SmSession.ObjectStore.ObjectFromData(ObjRec,True)

End Sub
```

A connected **SmObject** can be obtained in one of the following ways:

- **SmObjectStore.ObjectFromData**, with the second argument **Disconnect** set to False
- **SmObjects.Item** property
- Any other method that returns a single **SmObject** stored in another object. For an example, see **SmCompositeObject.Item** on page [99](#).

The lifetime of a connected object depends on the lifetime of the source object, meaning that **SmObject** becomes invalid after the source object is destroyed.

The mode of a **SmObject** object is indicated by its **CanRestructure** property.

Additional SmObject Functionality

Additional groups of **SmObject** functionality include the following:

- Object manipulation, **Insert, Update, Delete**
- Linking an object to other objects: **LinkOneLevel, LinkToParent, LinkToChild**
- Unlinking an object from other objects: **UnLinkParents, UnLinkChildren, UnLinkRelations, UnLinkChild, UnLinkParent, UnLinkRelation**
- Manipulating an object's related objects:
- Retrieve related objects - **RetrieveChildren, RetrieveParents, RetrieveRelations**
- Unlink and delete related objects from the **SmarTeam** database: **DeleteParents, DeleteChildren, DeleteRelations**
- Link related objects to another object, and, optionally, delete links with the current object: **MoveParentsToOtherObject, MoveChildrenToOtherObject**.

Note: These methods use the **SmQueryDefinition** object, as described on page [104](#).

Example

The following example sets a value in the CN_DESCRIPTION field of a specific object and updates it in the database:

```
Sub Test()  
  
    Dim SmObject as SmApplic.IsObject  
  
    Dim SmClass as SmApplic.SmClass  
  
    Set SmObject = SmSession.ObjectStore. NewObject(ClassId)  
  
    SmObject.ObjectId = ObjectId  
  
    SmObject.AddAttributes "CN_DESCRIPTION"  
  
    SmObject.Value("CN_DESCRIPTION") = "Updated motor engineering"  
  
    SmObject.Update  
  
End Sub
```


Example

This example links a specific object as a parent of another object:

```
Sub Test()  
  
    Dim ParObject As SmApplic.ISmObject  
  
    Dim ChildObject As SmApplic.ISmObject  
  
    Dim LinkAttributes As SmRecList.SmRecord  
  
    Dim HierClassId As Integer  
  
    Set LinkAttributes = New SmRecList.SmRecord  
  
    LinkAttributes.AddHeader "CN_QUANTITY", 2, sdtSmallInt  
  
    LinkAttributes.Value("CN_QUANTITY") = 3  
  
    HierClassId = ParObject.SmClass.DefaultHierarchicalClassId  
  
    ParObject.LinkToChild HierClassId, ChildObject, LinkAttributes  
  
End Sub
```

Example

The following example retrieves a specific object's children:

```
Sub Test()  
  
    Dim ParObject As SmApplic.ISmObject  
  
    Dim Children As SmApplic.ISmObjects  
  
    Dim QueryDefinition As SmApplic.ISmQueryDefinition  
  
    Dim Count As Integer, i As Integer  
  
    Dim SingleChild As SmApplic.ISmObject  
  
    'Retrieve children of the object  
  
    Set Children = ParObject.RetrieveChildren(QueryDefinition)  
  
    'Display primary identifier of each child  
  
    If Not (Children Is Nothing) Then  
  
        Count = Children.Count - 1  
  
        For i = 0 To Count
```

```
        Set SingleChild = Children.Item(i)

        MsgBox "Id of a child is " + SingleChild.Value("CN_ID")

    Next

End If

End Sub
```

When creating a new SmarTeam title object from either a GUI or API, the TDM_FILE_ID parameter is used. The internal attribute value is automatically assigned.

The following objects are used to assist in implementing the SmarTeam COM API parameter TDM_FILE_ID.

- Interface: ISmObject
Insert Method
Inserts object. Modification is executed according to DefaultBehavior object
Sub Insert()

Example:

```
Set NewObject = Session.ObjectStore.NewObject

Session.MetaInfo.SmClassByName("Solid EdgePart").ClassId

'Additional code if needed

NewObject.Insert
```

- Interface: ISmObject

Update Method

Updates object. Modification is executed according to DefaultBehavior object.

Sub Update()

Example

The following example demonstrates using the class SmApplic.ISmObject to perform operations and accessing data of Sm objects while updating object data in a Database.

```
Set NewObject = Session.ObjectStore.NewObject
(Session.MetaInfo.SmClassByName("Solid EdgePart").ClassId)

'.... Additional code if needed
```

```
NewObject.Insert
```

```
.
```

When updating an existing **SmObject** you should not hold the **TDM_FILE_ID** while it is updating object's record list.

For updating use the following example code:

```
Set SmObject = Session.ObjectStore.NewObject (Session.MetaInfo.SmClassByName
("Solid Edge Part").ClassId)

SmObject.ObjectId = NewObject.ObjectId

SmObject.AddAttributes
"TDM_ID;FILE_NAME;DIRECTORY;CAD_REF_DIRECTORY;CAD_REF_FILE_NAME;FILE_TYPE"

SmObject.Data.Value("TDM_ID") = Session.ObjectStore.Sequences.ItemByAttribute

(Session.MetaInfo.SmClassByName("Solid Edge Part").GetAttribute("TDM_ID",
True)).IncrementGlobal

SmObject.Data.Value("FILE_NAME") = "test.txt"

SmObject.Data.Value("DIRECTORY") = "C:\\"

SmObject.Data.Value("CAD_REF_DIRECTORY") = SmObject.Data.Value("DIRECTORY")

SmObject.Data.Value("CAD_REF_FILE_NAME") = SmObject.Data.Value("FILE_NAME")

SmObject.Data.Value("FILE_TYPE") =
Session.ObjectStore.GetSmLookUpByUniqueName(Session.MetaInfo.SmClassByName("Fi
le Type").ClassId, "Solid Edge Part").Id

SmObject.Update
```

SmObjects Object

The **SmObjects** object represents a collection of **SmObject** objects.

Depending on the method used to retrieve the **SmObjects** instance, it might not be possible to perform certain operations on it efficiently. The **CanRestructure** property indicates if the **SmObjects** instance can be restructured by adding or removing attributes from the objects it contains, and the **CanAddRemove** property indicates if objects can be added to the collection or removed from it.

Accessing SmObject

A **SmObject** object may be created using various methods, for example:

- **SmObjectStore.NewObjects** creates a new empty collection.
- **SmObjectStore.ObjectsFromData** passes an **SmRecordList** object as input, and obtains an **SmObjects** collection object that represents objects listed in the **SmRecordList** object.
- Any other method that enables an **SmObject** collection to be obtained.

SmBehavior Object

The **SmBehavior** object contains flags that determine whether specified conditions are executed.

All object management functions, such as **Insert**, **Update**, **Delete**, **Retrieve**, are executed according to behavior flags contained in the **SmBehavior** object. For example, when adding objects, you can set a flag to determine whether to:

- Execute scripts
- Prompt the user
- Check user authorization.

The **SmObjectStore** object contains a default **SmBehavior** object with default flag values.

All methods, such as **Add**, **Update**, **Delete**, and **Retrieve**, use the default behavior of **SmObjectStore**. Each of these methods has a counterpart method, which has the same name with the suffix **Ex**. These methods take a **SmBehavior** object as an argument.

To create your own **SmBehavior** object, use the **Clone** method and modify its flags, as shown in the following example:

Example

```
Sub test()  
  
    Dim SmSession As SmApplic.SmSession  
  
    Dim DocClassId as Integer  
  
    Dim DocObject As SmApplic.ISmObject
```

```
Dim NewBehavior As SmApplic.ISmBehavior

DocClassId = SmSession.MetaInfo.SmClassByName
("Document").ClassId

Set DocObject = SmSession.ObjectStore.NewObject(DocClassId)

DocObject.Value("CN_DESCRIPTION") = "doc1"

DocObject.Value("CN_ID") = "Doc-0010"

Set NewBehavior = SmSession.ObjectStore.DefaultBehavior.Clone

' Don't invoke scripts while performing operation
NewBehavior.InvokeScripts = False

' Don't prompt user to confirm operation
NewBehavior.ConfirmOperations = False

DocObject.InsertEx NewBehavior

End Sub
```

The settings for the default system behavior are as follows:

- CheckAuthorization = True
- InvokeScripts = True
- ConfirmOperation = coPromptUser
- CheckLinksOnDelete = cldPromptUser
- ConfirmAttachedFileDeletion = coPromptUser
- CheckObjectsExistence = True
- ViewObjectAuthorization = voaCheckAndDeleteFromList

The default behavior settings can be accessed through the **MetaInfo.ObjectStore.DefaultBehavior** property.

Authorization Settings

The two Behavior parameters:

CheckAuthorization

ViewObjectAuthorization

pertain to preventing an operation when the user does not have authorization privileges.

The CheckAuthorization parameter pertains to the Database operations Add, Update, Delete and has the values:

CheckAuthorization	Description
True	The operation is performed only if the user has SmarTeam authorization on the class on which the operation is running.
False	The operation is performed even if the user has no authorization the class.

The ViewObjectAuthorization parameter pertains to displaying the results of query operations and has the values:

ViewObjectAuthorization	Description
voaNotToCheck	All query results objects are returned, irrespective of user authorization.
voaCheckAndDeleteFromList	The query results contain only authorized objects.
voaCheckAndSignInList	All query results objects are returned, irrespective of user authorization, but any objects not authorized are not displayed by SmarTeam. The objects have a special signature in the RecordList.

SmMultiObjects Object

The **SmMultiObjects** object represents a collection of **SmObjects** and therefore the list may contain Persistent Objects of different superclasses.

The following example shows an iteration on an **SmMultiObjects** variable in order to obtain a specific **SmObject** object.

Example

```
Sub Test()  
  
    Dim SmObjsColl As SmApplic.ISmMultiObjects  
  
    Dim i As Integer, CollCount As Integer  
  
    Dim SingleColl As SmApplic.ISmObjects  
  
    Dim j As Integer, Col2Count As Integer  
  
    Dim SingleObj As SmApplic.ISmObject  
  
    CollCount = SmObjsColl.Count - 1  
  
    For i = 0 To CollCount  
  
        Set SingleColl = SmObjsColl.Item(i)  
  
        Col2Count = SingleColl.Count - 1  
  
        For j = 0 To Col2Count  
  
            Set SingleObj = SingleColl.Item(j)  
  
        Next j  
  
    Next i  
  
End Sub
```

SmCompositeObjects and SmCompositeObject Objects

The **SmCompositeObject** object represents a collection of **SmObject** objects that represent Persistent Objects related to each other by, for example, a hierarchical link or general link.

The **SmCompositeObjects** object represents a collection of **SmCompositeObject** objects that have the same composition information, for example, objects and their hierarchical links to a specific parent object.

Each participating object in the **SmCompositeObject** is identified by its specific key. This key may be either a character key or an integer key which represents the object's Class ID or superclass ID.

For example, the methods `SmObject.RetrieveChildrenAndLinks` and `SmObject.RetrieveParentsAndLinks` create an **SmCompositeObjects** collection object in which each **SmCompositeObject** member contains two **SmObject** objects.

These objects are identified as follows:

- The child object itself is identified by its superclass ID
- The link object is identified by its class ID.

In order to retrieve a specific **SmObject** object from a single **SmCompositeObject**, use the **Item** property. The input to the method can be either a character role or a class ID role, but not the index of the **SmObject** in the **SmCompositeObject**. To obtain a specific **SmObject** by its index, use one of the following:

```
Role = SmCompositeObject.Role(intIndex)
```

```
Set SmObject = SmCompositeObject.Item(Role)
```

or

```
ClassId = SmCompositeObject.ClassId(intIndex)
```

```
Set SmObject = SmCompositeObject.Item(ClassId)
```

The **SmMultiCompositeObjects** object represents a collection of **SmCompositeObjects** objects, and can therefore contain objects with different composition information.

SmLookupObjects and SmLookupObject Objects

These objects represent the Lookup Table information.

SmClassReferenceObjects and SmClassReferenceObject Objects

The **SmClassReferenceObject** object represents the value of an attribute that references an object from another class.

The **SmClassReferenceObjects** object provides access to the list of possible objects from the referenced class, which can be referenced by a specific attribute.

Managing Transactions in the Database

When working with Persistent Objects, it is often necessary to group several operations to be performed on the Persistent Objects into a transaction. Transactions are used to mark a group of operations as an “atomic operation”, meaning an operation that should either complete successfully or not at all.

The **SmarTeam** Object Model provides support for transactions through the **SmDatabaseConnection** object. However, the **SmarTeam** Object Model also provides the **Operation** model, which is often more useful for larger systems.

An operation is similar to a transaction. However, when using operations, you do not explicitly start, commit or rollback database transactions. Instead, you mark the beginning of an operation, and later, the successful, or unsuccessful, end of the operation. The **SmarTeam** database engine uses this information to manage database transactions in an efficient manner.

Unlike transactions, operations can be nested, meaning that you can start an operation while another is still in progress. However, the inner operation must end before the outer operation is completed. Nested operations are especially useful in larger systems, where a component needs to participate in a larger transaction without being aware of the exact scope of the transaction.

The following **SmObjectStore** methods manage operations:

- **StartOperation** is used to mark the beginning of an operation. If there is no active transaction at this point, the **SmarTeam** Engine begins a new transaction automatically. Each call to **StartOperation** must be matched with a later call to **EndOperation** or **FailOperation**.
- **EndOperation** is used to mark the successful completion of an operation. If the operation is a top-level operation (i.e. it is not nested in another operation) the **SmarTeam** Engine automatically commits the transaction at this point.
- **FailOperation** is used to mark a failed operation. When a nested operation fails, all the operations including it are also marked as failed, even if **EndOperation** is called for them. If the failed operation is a top-level operation, the **SmarTeam** Engine automatically rolls back the transaction at this point.

In the following example, the program is required to perform the **Update**, **Retrieve** and **LinkToParent** methods in one database transaction. **EndOperation** in this case performs the **COMMIT** transaction. If the methods fail, the **FailOperation** method accomplishes **ROLLBACK** of the whole transaction.

Example

```
Sub LinkObjects()  
    On Error GoTo ErrorHandler  
  
    StartOperation  
  
    Object.Update  
  
    Parent.Retrieve  
  
    Object.LinkToParent(LinkClassId, Parent)  
  
    EndOperation  
  
    ErrorHandler:  
  
    FailOperation  
  
End Sub
```

Below is an example of nested operations. In this example the overall transaction contains the method **UnlinkFromParent** and three methods from the **LinkObjects** function. **COMMIT** of the transaction is performed by the **EndOperation** from the main routine.

Example

```
Sub Main()  
    On Error GoTo ErrorHandler  
    StartOperation  
    Object.UnlinkFromParent(OldParent)  
    ' Call function from the previous example  
    LinkObjects  
EndOperation  
ErrorHandler:  
FailOperation  
...  
End main
```

SmQuery Object

Description

The **SmQuery** object allows you to define a powerful and flexible database query, run it, and access the query result.

The SmQuery object lets you perform the following advanced query tasks:

- Define class roles
- Perform a query on linked objects from the same class
- Order and sort the query results as desired
- Use query filters.

The following is a description of the important properties and methods of the SmQuery object:

- Query Definition
- QueryResult.

QueryDefinition

The **QueryDefinition** property provides access to the components of the query definition as follows:

- **Roles** – the collection of class-role assignments that participate in the query
- **Select** – the collection of class attributes which are returned in the query result
- **Where** – the collection of conditions that determine which objects will be included in the query result set
- **OrderBy** – the collection of attributes that define the sort order of the returned query result.

Roles

QueryRoles is the collection of class-role assignments on which the query performs its search. They are analogous to the tables in the FROM clause of an SQL statement.

A QueryRole item is a pair consisting of a class, represented by its class id, and a ClassRole, represented by one of the identifiers: “F”, “L”, and “S”. By assigning different ClassRoles to a class, the same class can assume different identities for the purpose of the query search.

The main application of the QueryRole is to allow queries on linked objects from the same class. The basic model is a first object, F, linked to a second object, S, by a link L (for that reason the letters “F”, “L” and “S” are used as ClassRole identifiers). For this model you need to define three QueryRole items in the query definition, with the “F”, “L”, and “S” identifiers, respectively.

To perform a query on a single class, define a single QueryRole item using the “F” identifier only. (For a single class query, the role can be whatever you want.)

As an example of the linked objects model, you might want to locate a document with certain attributes that has a child document with certain other attributes. The target document and its child document are in the same Document class and the parent-child link between them is in a third class, the Hierarchic Link class. In order to differentiate between parent and child attributes, the Document class is assigned two separate roles: the parent role and the child role (identifiers “F” and “S”, respectively). The Hierarchic Link class is assigned the link role “L”.

This allows you to specify Document class attributes for both parent and child documents in the **Select** collection and the **Where** collection of the query. You distinguish between them by appending the ClassRole identifier: F.attributename for the parent document and S.attributename for the child document.

For example: you want to find the name of a parent document whose name contains the string “automobile” that has a child Document that has more than 100 pages.

You define three QueryRole items as follows:

QueryRole Item	ClassRole	Class
Parent	F	Document Class
Link	L	Hierarchic Link Class
Son	S	Document Class

For example, the first row says that the QueryRole “Parent” is defined as the pair “F” and the class Document Class.

In terms of the **SmarTeam** notation, the ObjectID attribute of the Document class `TN_DOCUMENTATION` is called `OBJECT_ID` and the Parent and Son Object ID attributes in the Hierarchic Link class `DOCUMENT_TREE` are `PAR_OBJECT_ID` and `SON_OBJECT_ID`, respectively.

Then, using the above role definitions, the analogous SQL format of the above query would be:

```
SELECT F.name
FROM TN_DOCUMENTATION F, DOCUMENT_TREE L, TN_DOCUMENTATION S
WHERE F.name LIKE "%automobile%" AND
F.OBJECT_ID = L.PAR_OBJECT_ID AND S.OBJECT_ID = L.SON_OBJECT_ID
AND S.PAGECOUNT > 100
```

To add a `QueryRole` item to the collection, use the **Roles.Add** method. You specify the class ID and the associated `ClassRole` identifier.

Note that the identifiers “F” and “S” do not imply a specific order. They can be interchanged without affecting the result.

Specifying Select Attributes

Any attribute that you specify for the select part of the query must be identified with a specific `QueryRole` or class-role assignment, as discussed in the previous section. Thus when you specify a class attribute, you also need to specify the `ClassRole` identifier (“F”, “L” or “S”) associated with the attribute’s class.

You assign a select attribute to the query using the **Select.Add** method. The method provides both attribute name and `ClassRole` parameters.

For example,

```
QueryDef.Select.Add "CLASS_ID", "S", False
```

means that the query returns values of the `CLASS_ID` attribute of the class associated with the `ClassRole` S.

This is analogous to the SQL statement:

```
SELECT
```

```
S.CLASS_ID
```

If a class has been assigned two different roles, you can refer to the same class attribute twice in the query where each time you use a different role:

```
QueryDef.Select.Add "CLASS_ID", "F", False
```

```
QueryDef.Select.Add "CLASS_ID", "S", False
```

This is analogous to the SQL statement:

```
SELECT
```

```
F.CLASS_ID, S.CLASS_ID
```

Specifying Where Conditions

Any attribute that appears in a condition in the Where part of the query must be identified with a specific QueryRole or class-role assignment, as discussed above. Thus when you specify a class attribute, you also need to specify the ClassRole identifier (“F”, “L” or “S”) associated with the attribute’s class.

You assign a condition on an attribute using the **Where.Add** method. The method provides both attribute name and ClassRole parameters.

For example:

```
QueryDef.Where.Add "", "CN_VOLUME", "=", "1", False, "F"
```

specifies that the CN_VOLUME attribute of the class associated with ClassRole F equals 1.

This corresponds to the SQL statement WHERE section:

```
WHERE
```

```
F.CN_VOLUME = 1
```

If a class has been assigned two different ClassRoles, you can refer to the same attribute twice in the Where part of the query where each time you use a different ClassRole.

Ordering the Results

The **OrderBy** property allows you to specify the ordering of the query results by attribute.

The **OrderBy.Add** method provides two parameters by which you can specify two types of sort order:

- **SortOrder**
- **SortIndex**

SortOrder

For an individual role-attribute, you can use the **SortOrder** parameter to determine the ordering of the results according to that attribute's values as follows:

- **None**
- **Ascending**
- **Descending**

SortIndex

In addition, you can use the **SortIndex** parameter to specify a nested ordering of the results. The attribute with **SortIndex** = 1 is sorted first, according to its sort order. Any entries with the same attribute value are again sorted, now according to the attribute with **SortIndex** = 2, and so on.

For example, the attributes selected for a query are the name, year of manufacture and color of an automobile. If you want to sort all cars first alphabetically by name, then further sort all cars with the same name by year of manufacture and finally sort all cars with the same name and year by color you would set:

Attribute	SortOrder	SortIndex
Name	Ascending	1
Year	Ascending	2
Color	Ascending	3

SmarTeam Security

SmarTeam has a "data model level" security that allows the data model designer to specify permissions on data model entities. The security includes specifying permissions per class, as well invoking a script which allows implementation of security per object. The **ISmQuery** object supports SmarTeam security.

Note Because the **ISmSimpleQuery** object does not deal with classes or objects, and instead deals directly with SQL and **RecordList**, it is not aware of this mechanism and does not enforce it.

Object Diagram

The object diagram of ISmQuery is shown below:

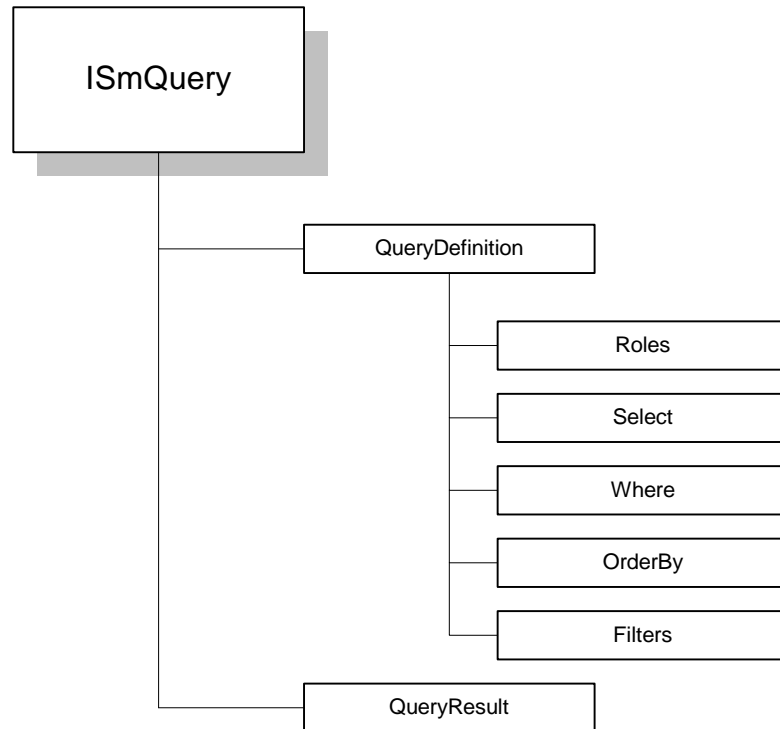


Figure 5-4 ISmQuery Object Diagram

Obtaining the ISmQuery Object

You obtain a Query object as follows:

```
Set SmQuery = SmSession.ObjectStore.NewQuery
```

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to a Query.

Query Task: Executing a Query

There are two methods of executing a query depending on how the query results are retrieved.

Execute a Query and Retrieve all Results – Run Method

Use the **Run** method to execute the query. The query operation is executed according to the DefaultBehavior object. All results of the query are placed in the QueryResults object.

See below for an example of how to use the Run method.

Use the **RunEx** method to execute the query according to a specified SmBehavior object.

Execute a Query and Retrieve one Result – Open Method

Use the **Open** method to open the query. This method executes the query according to DefaultBehavior object but instead of placing all the results in the QueryResults object – as the Run method does – only the first row of the QueryResults object is filled.

You would use the Open method instead of the Run method, for example, if you don't need to display all results at one time. You might use it for paging the results on an HTML display where the user can decide to view only part of the results.

See below for an example of how to use the Open method.

Use the **OpenEx** method to open a query where the query is executed according to a specified SmBehavior object.

Query Task: Defining the Query Mode

Use the **QueryMode** property to get or set the query mode.

The query mode specifies how the results of successive queries are handled in the results record list, as shown in the following table:

QueryMode	Description	Software Constant
Append Results	The results of the query are appended in the record list to the results of the previous query.	qmAppend
Build Results	All results of previous queries are deleted from record list before inserting the results of this query.	qmBuild
Intersect Results	The results of this query are intersected with the results of previous queries in the record list.	qmIntersect

Query Task: Defining a Query for One Class

This section presents an example of defining and performing a query on one class.

```
' Searching for all SolidWorks Assembly objects that have volume 1
'
' First get the class on which the query is to run
Set SWAssemblyClass = SmSession.MetaInfo.SmClassByName("SolidWorks Assembly")

' get a new query and query definition
Set SmQuery = SmSession.ObjectStore.NewQuery
Set QueryDef = SmQuery.QueryDefinition

' use one role, F, for one class
QueryDef.Roles.Add SWAssemblyClass.ClassId,"F"

' define SELECT attributes
QueryDef.Select.Add "OBJECT_ID","F",False
QueryDef.Select.Add "CLASS_ID","F",False

' define the WHERE conditions
```

```
QueryDef.Where.Add "", "CN_VOLUME", "=", "1", False, "F"  
  
' run it  
  
SmQuery.Run
```

Query Task: Defining a Query for Linked Objects

This section presents an example of defining and running a query on linked objects from the same class.

```
' Searching for all SolidWorks part objects that are children of any  
SolidWorks Assembly object with quantity > 4 and volume = 1  
  
'  
  
' First get the classes on which the query is to run  
  
' the object classes:  
  
Set SWAssemblyClass = SmSession.MetaInfo.SmClassByName("SolidWorks Assembly")  
  
Set SWPart = SmSession.MetaInfo.SmClassByName("SolidWorks Part")  
  
' and the link class  
  
HierClassId = SWPart.DefaultHierarchicalClassId  
  
' get a new query and query definition  
  
Set SmQuery = SmSession.ObjectStore.NewQuery  
  
Set QueryDef = SmQuery.QueryDefinition  
  
' define the F, L, and S roles  
  
QueryDef.Roles.Add SWAssemblyClass.ClassId, "F"  
  
QueryDef.Roles.Add SWPart.ClassId, "S"  
  
QueryDef.Roles.Add HierClassId, "L"  
  
' add the SELECT conditions  
  
QueryDef.Select.Add "NM_OBJECT_ID", "S", False  
  
QueryDef.Select.Add "NM_CLASS_ID", "S", False  
  
' add the WHERE conditions  
  
QueryDef.Where.Add "", "CN_VOLUME", "=", "1", False, "F"
```

```

QueryDef.Where.Add "", "CN_QUANTITY", ">", "4", False, "L"

' run it

SmQuery.Run

```

Query Task: Getting Query Results

Results can be extracted from the QueryResult object in different ways, depending on the types of QueryRoles that are defined in the Roles property, according to the following table:

QueryRoles Items	Object used to extract Results
Linked Classes	CompositeObjects
Single Class	ISmObject

See examples in the next section.

Examples

This section contains examples of the use of the Query object to define and execute queries.

Running a Query

This example uses the Run method on two linked classes. The results are displayed using a CompositeObject.

Example

' Searching for all SolidWorks Assembly and SolidWorks part objects that have parent-son relation where both have the status NEW

```

Sub Example(SmSession As SmApplic.SmSession)

    Dim SWAssemblyClass As SmApplic.ISmClass

    Dim SWPart As SmApplic.ISmClass

    Dim SmQuery As SmApplic.ISmQuery

    Dim QueryDef As SmApplic.ISmQueryDefinition

    Dim SmView As SmGUISrv.ISmView

    Dim GUIService As SmGUISrv.SmCommonGUI

    Dim LookUpObject As SmApplic.ISmLookUpObject

    Dim LookUpClassId As Integer

```

```
' First get the classes on which the query is to run
' the object classes:

Set SWAssemblyClass = SmSession.MetaInfo.SmClassByName("SolidWorks
Assembly")

Set SWPart = SmSession.MetaInfo.SmClassByName("SolidWorks Part")

' and the link class

HierClassId = SWPart.DefaultHierarchicalClassId

' get a new query and query definition

Set SmQuery = SmSession.ObjectStore.NewQuery

Set QueryDef = SmQuery.QueryDefinition

' define the F, L, and S ClassRoles

QueryDef.Roles.Add SWAssemblyClass.ClassId, "F"

QueryDef.Roles.Add SWPart.ClassId, "S"

QueryDef.Roles.Add HierClassId, "L"

' add the SELECT conditions for the SolidWorks part objects

QueryDef.Select.Add "OBJECT_ID", "S", False

QueryDef.Select.Add "CLASS_ID", "S", False

QueryDef.Select.Add "CN_DESCRIPTION", "S", False


' getting NEW State id

' first get id of lookup class of all states

LookUpClassId = SmSession.MetaInfo.SmClassByName("State").ClassId

' get NEW state item from lookup list of all states

Set LookUpObject =
SmSession.ObjectStore.GetLookupList(LookUpClassId).ItemByUniqueName("New")

' add the WHERE conditions

' Only Assemblies and Parts with status NEW will be retrieved
```

```

' STATE is field id from F or S classes; Lookupobject.id is id of NEW
state

QueryDef.Where.Add "", "STATE", "=", LookupObject.Id, False, "F"
QueryDef.Where.Add "", "STATE", "=", LookupObject.Id, False, "S"

' run it

SmQuery.Run

If (SmQuery.RecordCount > 0) Then 'found results

    ' Get main GUI service object

    Set GUIService = SmSession.GetService("SmGUISrv.SmCommonGUI")

    ' Create new SmView of type - bottom up tree)

    Set SmView = GUIService.Views.NewViewByType(vwtBottomUpTree)

    ' Assign displayed composite objects for view from query results

    ' Need CompositeObjects to display QueryResults of linked objects

    SmView.DisplayObjects.CompositeObjects =
SmSession.ObjectStore.CompositeObjectsFromData(SmQuery.QueryResult, False)

    ' Assign title to window

    SmView.ViewTitle = "Found: " & CStr(SmQuery.RecordCount) & " records"

    ' Show results by window object

    SmView.SmViewWindow.Show

Else

    MsgBox "No objects found for this query"

End If

End Sub

```

Opening a Query

This example uses the Open method on a single class. The results are displayed 20 records at a time using a CompositeObject.

Example

```

' Search for SolidWorks Assembly objects with status NEW

```

```
Sub Example(SmSession As SmApplic.SmSession)

    Dim SWAssemblyClass As SmApplic.ISmClass

    Dim SmQuery As SmApplic.ISmQuery

    Dim QueryDef As SmApplic.ISmQueryDefinition

    Dim SmView As SmGUIsrv.ISmView

    Dim GUIService As SmGUIsrv.SmCommonGUI

    Dim LookUpObject As SmApplic.ISmLookUpObject

    Dim LookUpClassId As Integer

    Dim Count As Long

    Dim Continue As Boolean

    Dim DisplCount As Integer

    ' define a query over one class

    ' First get the class on which the query is to run

    Set SWAssemblyClass = SmSession.MetaInfo.SmClassByName("SolidWorks
Assembly")

    ' get a new query and query definition

    Set SmQuery = SmSession.ObjectStore.NewQuery

    Set QueryDef = SmQuery.QueryDefinition

    ' define the F role

    QueryDef.Roles.Add SWAssemblyClass.ClassId, "F"

    ' add the WHERE conditions

    ' get Lookup class ID for internal state class

    LookUpClassId = SmSession.MetaInfo.SmClassByName("State").ClassId

    ' get NEW state item from states lookup list

    Set LookUpObject =
SmSession.ObjectStore.GetLookupList(LookUpClassId).ItemByUniqueName("New")
```



```
' Only Assemblies with status NEW will be retrieved
' STATE is a field id from F ; Lookupobject.id is id of state NEW
QueryDef.Where.Add "", "STATE", "=", LookUpObject.Id, False, "F"
' add description
QueryDef.Select.Add "CN_DESCRIPTION", "F", False
' Open query
SmQuery.Open
' Get main GUI service object
Set GUIService = SmSession.GetService("SmGUISrv.SmCommonGUI")

Count = 0      ' total QueryResults record count
Continue = True

' if not at end of QueryResult and want to continue to display
While ((Not SmQuery.EOF) And (Continue))

    DisplCount = 0      ' count of records displayed on window

    ' Create new SmView of type - tree list to show founds object and
    ' their sons
    Set SmView = GUIService.Views.NewViewByType(vwtTreeList)
    SmView.ViewTitle = "Push Ok button to show next 20 records found "

    ' Assign displayed objects for view from query results
    While ((Not SmQuery.EOF) And (DisplCount < 20))

        DisplCount = DisplCount + 1

        Count = Count + 1

        ' add one object to display from QueryResult
        ' (CompositeObjectFromData gets one item)

        SmView.DisplayObjects.CompositeObjects.Add
        SmSession.ObjectStore.CompositeObjectFromData(SmQuery.QueryResult, 0, True)

    ' retrieve next QueryResult record
```

```
        SmQuery.Next
    Wend

    If DisplCount > 0 Then
        ' display data/dialog window
        SmView.SmViewWindow.ShowModal

        ' user pressed Cancel, finish displaying
        If SmView.SmViewWindow.ModalResult = mrCancel Then
            Continue = False
        End If
    End If

Wend

If Count = 0 Then
    MsgBox "No objects found for this query"
End If

' Close Query
SmQuery.Close

End Sub
```

ISmSimpleQuery function

The ISmSimpleQuery function enables the execution of a raw SQL query. However, while allocating as much memory as necessary to generate results for the query, the ISmSimpleQuery function does not monitor memory overflow.

Therefore, when performing a query on a large amount of data, it is recommended that you use SmIncrementalSimpleQuery, which uses paging and does not cause memory overflow.

6. SmarTeam GUI Services Library

General Description

The **SmarTeam** GUI Services library comprises objects that enable the following **SmarTeam** Windows client-related functionality:

- Create and display **SmarTeam** views
- Retrieve information about existing GUI components
- Display various **SmarTeam** windows and dialogs.

Dependencies

The **SmarTeam** GUI Services library has the following dependencies:

- **SmarTeam** Record List library
- **SmarTeam** Engine library.

GUI Concepts

This section describes the basic GUI concepts used in the **SmarTeam** GUI services.

The SmarTeam View

The **SmarTeam** View is a window with a standard and consistent design, layout and operation that is used to display a variety of persistent SmarTeam objects and links in various formats. You can also get selected objects from the display.

SmarTeam Dialogs

SmarTeam provides a series of standard Dialogs for inputting and selecting information such as Local Files Explorer and Open Dialog or for performing operations such as Login and Save As.

Overview of Objects—ISmCommonGUI

This section presents an overview of the main ISmCommonGUI objects including a description of the associated objects that are useful for the programmer:

The ISmCommonGUI is the highest-level object; its main purpose is to contain the other objects.

The major ISmCommonGUI components are shown in the following object diagram:

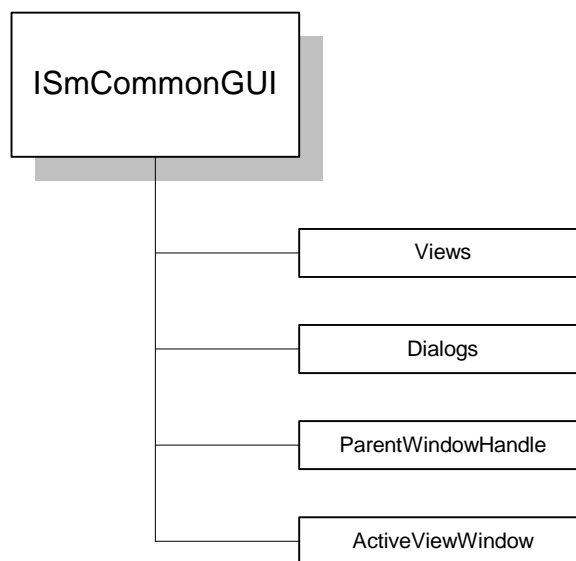


Figure 6-1 ISmCommonGUI Object Diagram

Properties

The CommonGUI object contains the following properties:

Property	Description
Views	Includes a set of methods for creating new SmarTeam View and a collection of currently existing SmarTeam View Window. Returns ISmViews.
Dialogs	Includes a set of methods for displaying the SmarTeam standard dialogs and windows. Returns ISmCommonDialogs.
ParentWindowHandle	The handle of the window that serves as a parent window for the windows and dialogs displayed by the SmarTeam API.
ActiveViewWindow	Accesses the currently active SmarTeam View Window. Returns ISmViewWindow.

Obtaining the ISmCommonGUI Objects

You can access the GUI Services through the SmGUIsrv.SmCommonGUI SmarTeam Service object, which is accessible through the GetService function of the Session. The ProgId of this object is: SmGUIsrv.SmCommonGUI.

A CommonGUI object is obtained as follows:

```
Dim CommonGUI As SmGUIsrv.SmCommonGUI
```

```
Set CommonGUI = SmSession.GetService('SmGUIsrv.SmCommonGUI')
```

The Views Property

The Views property provides a set of methods for creating new **SmarTeam** Views and contains a collection of currently existing **SmarTeam** View Windows.

Properties

The ISmViews object has the following properties:

Property	Description
Windows	The collection of currently existing SmarTeam ViewWindow object. Returns ISmViewWindows. See the ISmView object for a description of the ISmViewWindow object.

Methods

The ISmViews object has the following methods:

Method	Description
NewViewByType	Creates a new SmView instance of the specified ViewType (see Table 6-1). Returns ISmView.
NewViewByName	Creates a new instance of a named SmView. Named View definitions are created by the user in various SmarTeam applications and stored in the Query and View subsystem of SmarTeam. Returns ISmView.
NewLifeCycleView	Creates a new life cycle SmView instance. Returns ISmLifeCycleView

Example

Use the NewLifeCycleView method to create a new Life Cycle view.

```
Set SmView = CommonGUI.Views.NewLifeCycleView
```

ISmView

The ISmView object represents the design, contents, operations and layout of a **SmarTeam** View. See the ISmViewWindow object for the physical characteristics of a **SmarTeam** View.

An ISmView object has the following characteristics:

Layout – A **SmarTeam** View is a two-sided display: the left side, the controller, displays objects and the right side displays the object data for an object selected on the controller. The highest-level object displayed on the controller is referred to as the leading object.

Standard views – you can create standard **SmarTeam** views for displaying different types of data including, for example, Parent-Child Tree, Custom and General Links. The complete set of standard views available is listed in [Table 6-1](#). You can also create a user-defined view and a life-cycle view (see “Obtaining the ISmView Object” below.)

Controller format – the format of the controller – either a tree or a grid – is determined by the choice of standard view.

Contents – The type of data that can be displayed on the View and the way it is loaded is determined by the choice of standard view.

Sample Standard Views

[Figure 6-2](#) shows a MainClassTree **SmarTeam** View. The controller component is a tree, and the right side shows a profile card component for the object selected on the controller.

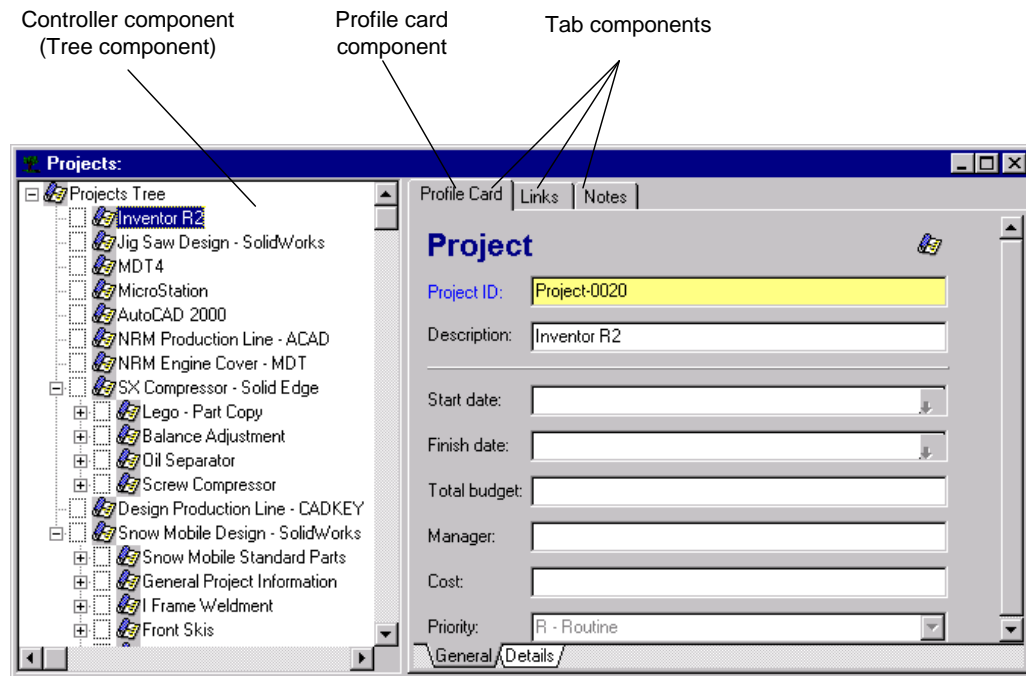


Figure 6-2 MainClass View with Tree Controller

[Figure 6-3](#) shows a WhereUsed **SmarTeam** View where now the controller is a grid and the leading object is displayed.

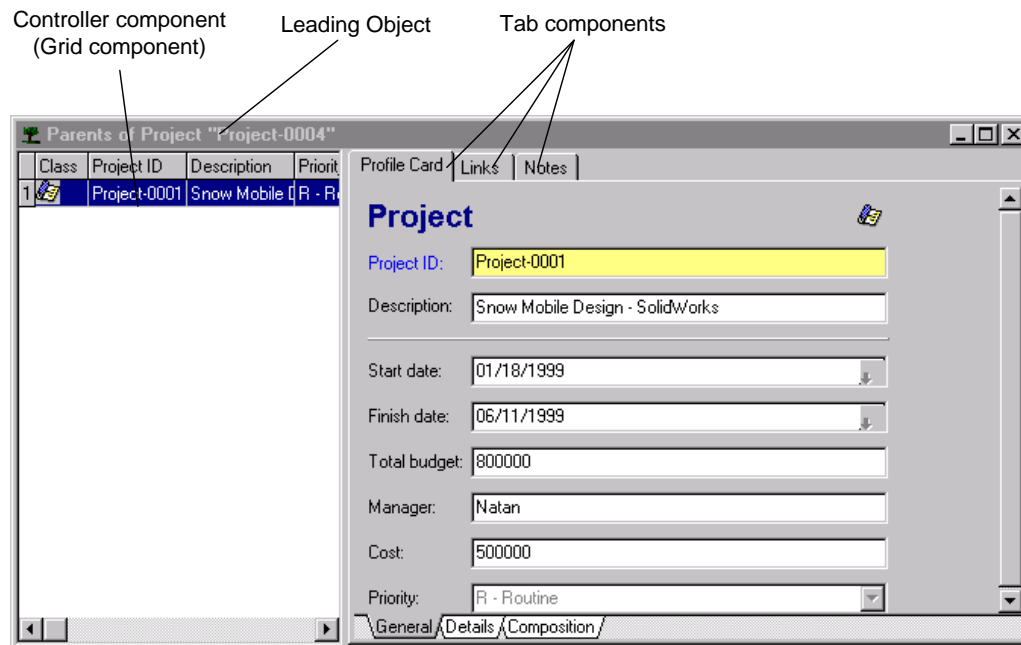


Figure 6-3 WhereUsed View with Grid Controller

Object Diagram

The object diagram of ISmView is shown below:

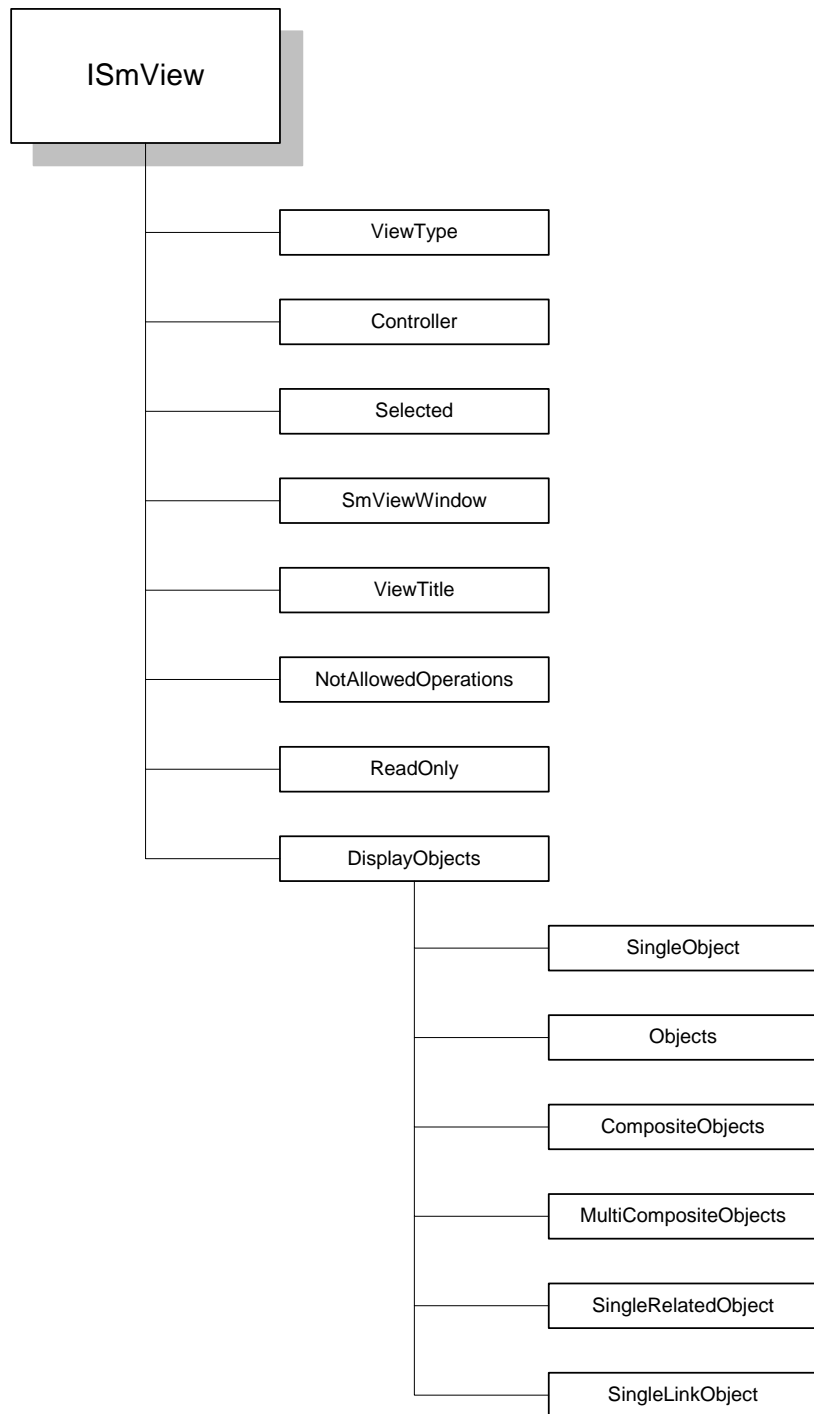


Figure 6-4 *ISmView Object Diagram*

Properties

The ISmView object has the following properties:

Property	Description
ActiveComponent	Returns an SmGUIComponent object representing the active component on the SmarTeam view.
Controller	Returns an SmGUIComponent object representing the controller object of the SmarTeam view.
NotAllowedOperations	Returns a SmOperations object representing operations that are not allowed on this View. User can fill in this list before calling Show method of the corresponding SmViewWindow object.
Components	Collection of all components of the SmarTeam view
ReadOnly	If true, this view is opened as read-only
StdContexts	Represents the button set for this view (for internal use)
Contexts	Sets the button set for this view (for internal use)
Selected	Returns an SmComponentObjects object representing the persistent objects selected on the View.
SmViewWindow	Returns and sets the SmViewWindow object corresponding to the SmView.
ViewIdentifier	The identifier of the View in the SmarTeam database. It can be the name of the search or the name of the view.
ViewTitle	Returns and sets the View title.
ViewType	Returns the View type (see Table 6-1)
ProductViewId	Sets or returns the Id of the Product View
DisplayObjects	Returns and sets SmComponentObjects, which represents persistent objects that are displayed on the controller of the View. This property is relevant to Views with the ViewType <ul style="list-style-type: none"> • vwtSingleObject • vwtTreeList • vwtCustom
ISmComponentObjects	
SingleObject	Returns or sets a SmObject object.
Objects	Returns a SmObjects object.
CompositeObjects	Returns or sets a SmCompositeObjects object.
MultiCompositeObjects	Returns or sets a SmMultiCompositeObjects object.
SingleRelatedObject	
SingleLinkObject	

Methods

The ISmView object has the following methods:

Method	Description
--------	-------------

Close	Closes view.
Refresh	Refreshes objects displayed on the view.
RefreshOperationIcon	Refreshes operation icon (relevant for Life Cycle views).

View Types

Table 6-1 presents a list of View Types available for creating standard Views. See Table 6-3, for the settings required for each View Type.

Table 6-1 View Types

View Type	The Controller Component shows:	Software Constant
ParentChildTree	Parent-child tree for single object	vwtParentChildTree
Custom	Grid	vwtCustom
GeneralLinks	List of objects linked to the leading object	vwtGeneralLinks
Revisions	All revisions of the leading object	vwtRevisions
SingleObject	A profile card for a single object. The profile card has a navigator.	vwtSingleObject
WhereUsed	Parents of the leading object	vwtWhereUsed
ComposedOf	A list of children of the leading objects	vwtComposedOf
TreeList	A parent-child tree (can come from several objects).	vwtTreeList
BottomUpTree	A child-parent tree (reverse of parent-child)	vwtBottomUpTree
MainClassTree	A list of persistent objects from a specific class, which are linked to the leading object arranged as a tree. Typically, the leading object is the object from the main class in the Demo database "Project".	vwtMainClassTree
ProductViewTree	Product view tree	vwProductViewTree

Obtaining the ISmView Object

As mentioned, you can create three different categories of views, using the methods of the ISmViews object as follows:

View Category	Method
Create a standard, pre-defined SmarTeam View. Each view type is appropriate for presenting a different aspect of the system (see Table 6-1). Returns ISmView.	NewViewByType
Creates a user-defined SmarTeam View, where the view has been created and named by the user of a SmarTeam application and has been stored in the Query and View subsystem of SmarTeam. The new instance is created according to the name given to the View by the user. Returns ISmView.	NewViewByName
Creates a new life cycle SmView instance. Returns mLifeCycleView	NewLifeCycleView

Example

The following creates a standard, pre-defined View of type Top-Down Tree list and displays a tree browser showing each object from the attached Search as a root, and the lower level classes for each of these roots.data on it:

```
Set SmView = CommonGUI.Views.NewViewByType(vwtTreeList)

' Get ISmViewWindow object attached to ISmView object

Set SmViewWindow = SmView.SmViewWindow

' Set collection of objects in view

SmView.DisplayObjects.CompositeObjects =
Session.ObjectStore.CompositeObjectsFromData(WorkObjects.Data, False)

' Set View Title

SmView.ViewTitle = "Selected object with children"

' Set window style as MDI child Sm window

SmViewWindow.Style = swsMDIChild

' Show view window

SmViewWindow.Show
```

ISmViewWindow

Description

The **ISmViewWindow** object represents the physical attributes of a **SmarTeam** View. It is associated with a **ISmView** object.

The object diagram of **ISmViewWindows** is shown below:

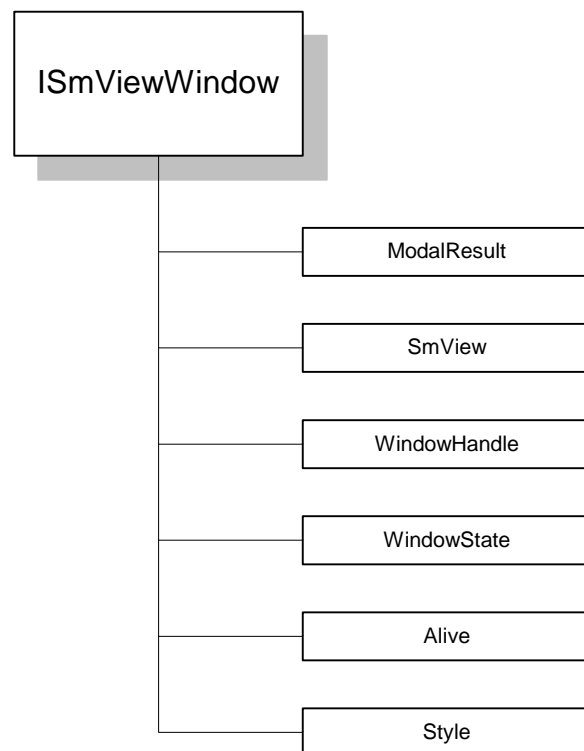


Figure 6-5 *ISmViewWindow* Object Diagram

Properties

The ISmViewWindow object has the following properties:

Property	Description
Alive	True if window is still alive (hasn't been closed)
ModalResult	Returns modal result of the windows opened in the modal m See Table 6-2.
SmView	Returns or sets an SmView object representing the corresponding SmarTeam view.
WindowHandle	The window handle
WindowState	Returns or sets the state of the window
Style	Returns or sets style of the window. swsNormal swsMDIChild Note: You can use the swsMDIChild style only when opening a V in the SmarTeam application and not, for example, from an integration.

Methods

The ISmViewWindow object has the following methods:

Method	Description
BringToFront	Brings window to front.
Close	Closes the window.
Show	Displays the window in regular mode
ShowModal	Displays window in modal mode.

Modal Result

Modal Result is relevant when you display the View using the ShowModal method. Modal views are used to elicit a user response about the objects displayed in the view. While a modal view is displayed, the focus is on it alone. Control returns to the main window only when the modal view is closed.

[Table 6-2](#) shows the possible modal results.

Table 6-2 Modal Results

Button/Action	ModalResultValue
OK	mrOK
Cancel	mrCancel
Help	mrHelp
Yes	mrYes

No	mrNo
Close	mrClose
Abort	mrAbort
Retry	mrRetry
Ignore	mrIgnore
All	mrAll
NoAll	mrNoAll

Obtaining the ISmViewWindow Object

A ViewWindow object is obtained from SmView as follows:

```
Set ViewWindow = SmView.ViewWindow
```

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to a ViewWindow and its components.

ViewWindow Task: Displaying the Window

Example

The following example shows how to open a **SmarTeam** view defined in the **SmarTeam** database by name and display it. The identifier of the view in the database is "All SolidWorks Assembly"

```
Set View = CommonGUI.Views.NewViewByName("All SolidWorks Assembly")  
  
View.SmViewWindow.Show
```

ViewWindow Task: Get the Modal Window Result

Use the **ModalResult** property to determine the button or action that the user used to close the form. The ModalResult property defines the result of the modal dialog.

Example

This example shows how to use the ShowModal and ModalResult properties.

```
Set View = CommonGUI.Views.NewViewByName("All SolidWorks Assembly")
```

```
View.SmViewWindow.ShowModal  
  
If View.SmViewWindow.ModalResult = mrOK Then  
    . . .  
End If
```

ViewWindow Task: Get the Window State

Use the WindowState property to get or set the display state of a window.
The possible states are:

WindowState	Software Constant
Normal	wstNormal
Minimized	wstMinimized
Maximized	wstMaximized

ISmGUIComponent

A **SmarTeam** View is composed of components, which are represented by ISmGUIComponent objects. The collection of all ISmGUIComponent objects is represented by ISmGUIComponents.

Properties

The ISmGUIComponent object has the following properties:

Property	Description
Name	Returns a name of the GUI component.
ComponentType	Returns a type of the GUI component. <ul style="list-style-type: none"> • Tree • Grid • TreeList • ProfileCard • ToolBar • Menu • Control
Visible	True if the GUI component is set as visible.
Enabled	True if the GUI component is set enabled.
ReadOnly	True if the GUI component is set Read Only.

Specifying the Controller GUI Component

The controller GUI component of a **SmarTeam** View (described under the section ISmView) is associated with one of the following ISmGUIComponent types:

- Tree
- Grid

The controller GUI component assumes one of these types when you create the SmView object, and depends on the value of the ViewType parameter that you specify. Table 6-3 indicates the component type for each ViewType.

Two ISmGUIComponent objects are provided to support working with the controller as a separate object:

- ISmTreeComponent
- ISmGridComponent

The controller object is defined and used as follows, for example:

```
Dim Controller As ISmTreeComponent
```

```
Set View = CommonGUI.Views.NewViewByType(vwtParentChildTree)
```

```
Set Controller = View.Controller
```

When the View object is created as a standard tree type of display (vwtParentChildTree), its Controller property is automatically set to a ISmTreeComponent type. Therefore the Controller object is defined with that type.

ISmTreeComponent and ISmGridComponent

As mentioned, the controller object assumes one of the two objects ISmTreeComponent and ISmGridComponent,, depending on the ViewType used when creating the View with which the controller is associated.

The object diagram of ISmTreeComponent and ISmGridComponent is shown below:

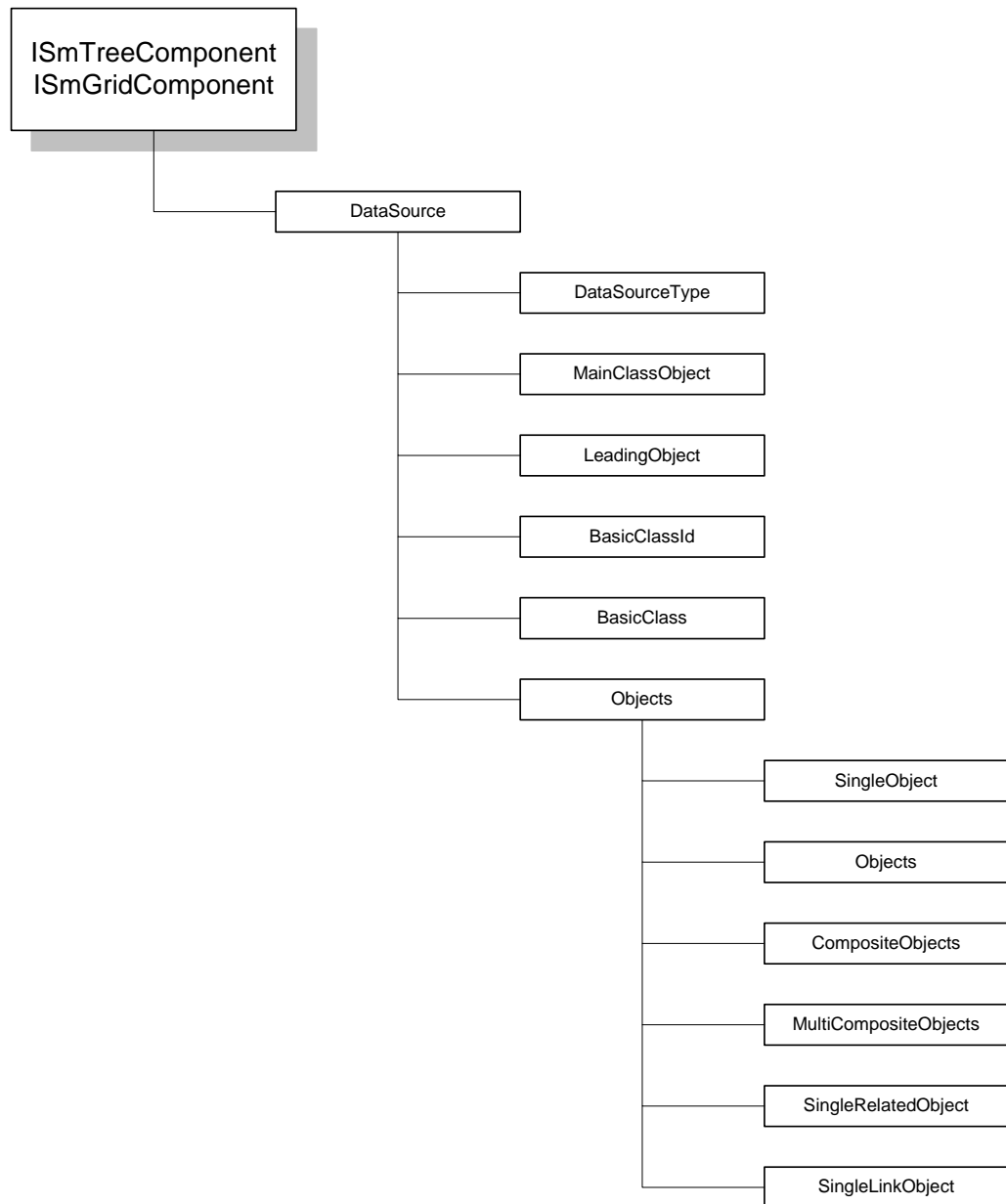


Figure 6-6 *ISmTreeComponent/ISmGridComponent* Object Diagram

Properties

The ISmTreeComponent/ ISmGridComponent object has the following properties and sub-properties:

Property	Description
DataSource	Returns an object of type of ISmDataSource.
ISmDataSource	
DataSourceType	Returns and sets type of data.
MainClassObject	Returns or sets an SmObject object representing data source object from the main class of the data model.
LeadingObject	Returns or sets an SmObject object representing leading object of the data source.
BasicClassId	Returns or sets basic class of the data source.
BasicClass	Returns an SmClass object representing basic class of the data source.
DataSourceType	Returns and sets type of data.
Objects	Returns an object of the type ISmComponentObjects.
ISmComponentObjects	
SingleObject	Returns or sets an SmObject object.
Objects	Returns an SmObjects object.
CompositeObjects	Returns or sets an SmCompositeObjects object.
MultiCompositeObjects	Returns or sets an SmMultiCompositeObjects object.
SingleRelatedObject	
SingleLinkObject	

Methods

The ISmTreeComponent/ISmGridComponent object has the following methods:

Method	Description
AddObjects	Not implemented.
DeleteObjects	Not implemented
UpdateObjects	Not implemented.
Select	Not implemented
SelectInTree	Not implemented.
GetSelected	Not implemented

Example

This example displays the general links for an object

```
Dim GridComponent as SmGUI.Srv.ISmGridComponent

Set GridComponent = SmView.Controller

GridComponent.DataSource.LeadingObject = <Leading object of the view>

GridComponent.DataSource.BasicClassId = <Class identifier of the objects
related to the leading objects to be shown on the controller>
```

Specifying Contents for a Standard View

The way you specify the contents to be displayed on a standard View depends on the particular View Type you are using.

There are two general ways to specify content for a View:

- Use the DisplayObjects property of the View object
- Use the DataSource property of the Controller object

Where the way you use depends on which View Type you specify.

Table 6-3 shows the settings required to specify the contents of a View for each standard View Type.

Table 6-3 Loading View Contents According to View Type

View Type	Description
Single Object vwSingleObject	Displays one entry field in which the user enters the name part of a name) of an object to display its Profile Card
Settings: SmView.DisplayObjects.SingleObject = <Single SmObject of persistent object>	

Top-Down Tree list vwtTreeList	Displays a tree browser showing each object from the attached Search as a root, and the lower level classes for each of the roots.
Settings: SmView.DisplayObjects.CompositeObjects = <CompositeObjects Collection of Parents>	
Ordinary (Custom) vwtCustom	Display an ordinary list of objects (not hierarchically)
Settings: 1) Display list of persistent objects SmView.DisplayObjects.CompositeObjects = <Collection SmCompositeObjects of persistent objects> For example: SmView.DisplayObjects.CompositeObjects = Session.ObjectStore.CompositeObjectsFromData(WorkObjects.Data, False) Note: In order to show a collection of the persistent objects by the DisplayObjects property you need to convert it to an object of type CompositeObjects or MultiObjects. 2) Display named view No preliminary settings needed	
Revisions vwtRevisions	Display a list of revisions
Where-Used List vwtWhereUsed	Display a list of all the parents of the selected object
Composed-Of List vwtComposedOf	Display the list of all the children of the selected object
Settings: Dim GridComponent as SmGUISrv.IsmGridComponent Set GridComponent = SmView.Controller GridComponent.DataSource.LeadingObject = <Leading object of the view>	
General Links vwtGeneralLinks	Display a list general links.
Settings: Dim GridComponent as SmGUISrv.IsmGridComponent Set GridComponent = SmView.Controller GridComponent.DataSource.LeadingObject = <Leading object of the view> GridComponent.DataSource.BasicClassId = <Class identifier of the objects related to the leading objects to be shown on the controller>	
Top-Down Tree vwtParentChildTree	Display a tree browser showing the selected object or the first object from the attached Search as the root, and its lower level classes
Bottom-Up Tree vwtBottomUpTree	Displays a tree browser that is a hierarchical display of objects according to the selected object.

Settings:

Dim TreeComponent as SmGUI.Srv.ISmTreeComponent

Set TreeComponent = SmView.Controller

1) Display desktop objects from the specific class related to specific Main Class object:

```
TreeComponent.DataSource.MainClassObject = <Project Object>
```

```
TreeComponent.DataSource.BasicClassId = <Class identifier of objects linked  
to Project Object to be shown on the controller >
```

2) Display child objects related to one leading object:

```
TreeComponent.DataSource.LeadObject = <Leading object of the view>
```

MainClassTree	
vwtMainClassTree	Top-down tree view of the main class objects
No preliminary settings	

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to a View.

View Task:

Getting Selected Objects

Use the **Selected** property to get the objects that were selected on the view.

```
' Show view window
```

```
SmViewWindow.Show
```

```
MsgBox "Select object to change description"
```

```
' Get collection of selected objects in view
```

```
Set SelectedObjects = SmView.Selected.Objects
```

**View Task:
Refresh the View.**

Use the Refresh method to refresh objects displayed on the View.

The object attributes needed to be refreshed can come from either the collection object itself in memory or – if the information is not in memory – from the database. Set the RetrieveFromDatabase parameter equal to “False” to get the information from memory. Otherwise, set the RetrieveFromDatabase parameter equal to “True”.

Note: Be sure that you actually need to retrieve object information from the Database for the refresh operation. Excessive retrieval of information from the Database is time consuming and may affect the performance of **SmarTeam**.

You can specify the following refresh actions:

Refresh Action	Description	Software Constant
Add	Add an object to the view	rfaAdd
Update	Update the objects specified	rfaUpdate
Delete	Delete the objects specified	rfaDelete

Example

This example refreshes the selected objects on the view. The object details are taken directly from the RefreshObjects collection without accessing the database.

```
RetrieveFromDatabase = False
```

```
SmView.Refresh rfaUpdate, RefreshObjects, RetrieveFromDatabase
```

ISmActiveWindow

The ISmActiveWindow object represents the ISmViewWindow that is currently active (see the section ISmViewWindow for details).

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to a ISmActiveWindow.

**CommonGUI Task:
Get the Active Window and Active View.**

Use the **ActiveViewWindow** property to get the active window and through it, the active view properties.

```
' Get CommonGUI object from Sm session

Set CommonGUI = Session.GetService("SmGUISrv.Sm SmCommonGUI")

' Get active smarteam view window

Set ViewWindow = CommonGUI.ActiveViewWindow

' Get active smarteam view

Set View = ViewWindow.SmView

' Show view title

MsgBox "View title: " & View.ViewTitle
```

Using ISmView and ISmViewWindow

This section shows an example using the GUI objects.

```
' This script demonstrates using ISmView and ISmViewWindow classes

' for GUI operations with SmarTeam

Sub TestName(Session As SmApplic.SmSession, WorkObjects As
SmApplic.ISmObjects)

    Dim CommonGUI As SmGUISrv.SmCommonGUI ' main object for creating new views

    Dim SmView As SmGUISrv.ISmView ' view properties and data

    Dim SmViewWindow As SmGUISrv.ISmViewWindow ' window methods

    Dim SelectedObjects As SmApplic.ISmObjects ' collection of selected
objects

    Dim SmObject As SmApplic.ISmObject ' first selected object

    Dim RetrieveFromDatabase As Boolean

    ' Retrieve GUI service object from Sm session
```

```
Set CommonGUI = Session.GetService("SmGUISrv.SmCommonGUI")

' Create new Sm view - using SmCommonGUI service creates a new
ISmViewWindow automatically

Set SmView = CommonGUI.Views.NewViewByType(vwtTreeList)

' Get ISmViewWindow object attached to ISmView object

Set SmViewWindow = SmView.SmViewWindow

' Set collection of objects in view

SmView.DisplayObjects.CompositeObjects =
Session.ObjectStore.CompositeObjectsFromData(WorkObjects.Data, False)

' Set View Title

SmView.ViewTitle = "Selected object with children"

' Set window style as MDI child Sm window

SmViewWindow.Style = swsMDIChild

' Show view window

SmViewWindow.Show

MsgBox "Select object to change description"

' Get collection of selected objects in view

Set SelectedObjects = SmView.Selected.Objects

Set SmObject = SelectedObjects(0).Clone

' Get current attribute value, add word and update object

OriginalValue = SmObject.Data.ValueAsString("CN_DESCRIPTION")

SmObject.Data.ValueAsString("CN_DESCRIPTION") = "Test" & OriginalValue

SmObject.Update

' Refresh view according to Database objects' attribute values

RetrieveFromDatabase = True

' Refresh view - update view of selected objects

SmView.Refresh rfaUpdate, SelectedObjects, RetrieveFromDatabase

End Sub
```

ISmDialogs

The ISmDialogs object includes a set of methods for creating new **SmarTeam** Dialogs.

Methods

The ISmDialogs object has the following methods:

Method	Description
NewLocalFilesExplorer	Creates Local Files Explorer dialog. Returns ISmLocalFilesExplorer. This dialog is accessed from SmarTeam through Tools/Local File Explorer.
NewSaveAsDialog	Creates Save As dialog. Returns IISmSaveAsDialog. Used, for example, to save an object from a Integration
NewOpenDialog	Creates Open dialog. Returns ISmOpenDialog. Used, for example, to insert objects in a Integration.
ExecuteLogin	Displays the SmarTeam login screen and executes log database.
ExecuteSelectClass	Display class structure tree for all classes in the database and allows user to select a class.
ExecuteSelectDatabase	Displays all databases available and allows user to select a database or to add or remove a database from the list. This dialog is accessed from SmarTeam through File/Switch to Database
ExecuteUserPreferences	Displays the SmarTeam user preferences and allows user to change preferences. This dialog is accessed from SmarTeam through Tools/Options
ExecuteAdminPreferences	Displays the SmarTeam administrative preferences and allows administrator to change preferences. This dialog is accessed from SmarTeam through Tool/Administrator Options
ExecuteSelectFromQueryResult	Opens the query editor, enabling the user to run a specified query and select objects. The selected objects reside in the function's return value. This dialog is accessed from SmarTeam through the "Find Object" icon on SmarTeam toolbar
ExecuteVaultMaintenance	Displays screen for managing Vault. This dialog is accessed from SmarTeam through Tools/Vault Maintenance.
ExecuteQueryByAttributes	Displays "Query By Attribute" dialog, returns modal result "OK" and ViewWindow object, if user performs query.

	query. This dialog is accessed from SmarTeam through the "Find Object by Attributes" icon on the SmarTeam toolbar
ExecuteQueryByExample	Displays "Query By Example" dialog, returns modal result "OK" and ViewWindow object, if user performs query This dialog is accessed from SmarTeam through "Find Object by Example" on the SmarTeam toolbar.
ExecuteQueryEditor	Displays "Query Editor" dialog, returns modal result "OK" and ViewWindow object, if user performs query.
ControlProperties	Enables the programmer to alter the appearance of the Save As dialog box.
OptionsProperties	Enables the Options by changing the values in the Save Options dialog box

Basic Dialogs

The following sections describe some of the basic dialogs that can be created using the ISmDialogs object:

- Select Database Dialog
- Select Class Dialog
- Select from Query Dialog
- Query By Attribute Dialog

Select Database Dialog

Use this dialog to choose between the databases available on the system.

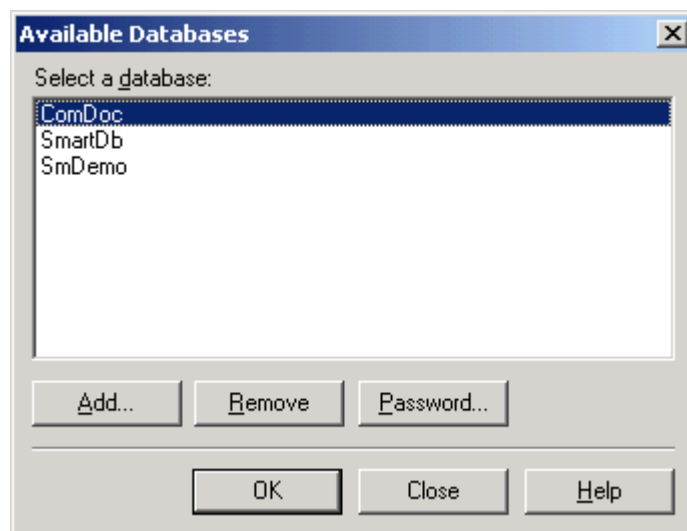


Figure 6-7 Select Database Dialog

Example

The following code produces the dialog in Figure 6-7.

```
Dim DatabaseName as Variant  
  
CommonGUI.Dialogs.ExecuteSelectDatabase DatabaseName  
  
MsgBox DatabaseName
```

Select Class Dialog

Use this dialog to choose one of the classes available.

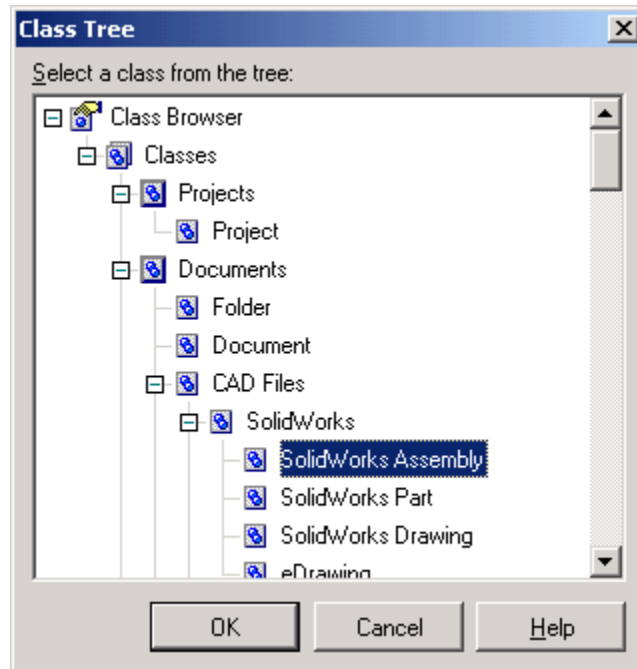


Figure 6-8 Select Class Dialog

Example

The following code produces the dialog in Figure 6-8.

```
Dim ClassId as Integer  
  
CommonGUI.Dialogs.ExecuteSelectClass 1, ClassId  
  
MsgBox ClassId
```

ExecuteSelectFromQueryResult Dialog

Use this method to:

- Display the list of predefined queries

- Run a predefined query and display the query results

- Select objects from the query results and return them

- Use the Query Editor to define a new query

Save the query as a predefined query

It is called in the form:

```
ModalResult = ExecuteSelectFromQueryResult(MultiObjects As ISmMultiObjects)
```

The selected objects are returned in the MultiObjects argument.

Example:

```
Dim MultiQueryResult As SmApplic.ISmMultiObjects
Dim Objects As SmApplic.ISmObjects
Dim Library As SmApplic.ISmObject

ModalResult = ExecuteSelectFromQueryResult(MultiQueryResult)
Set Objects = MultiQueryResult.Item(0)
Set Library = Objects.Item(0)

MsgBox CStr(Library.Value("OBJECT_ID"))
```

ExecuteQueryByAttributes Dialog

This method displays the "Find Object By Attribute" dialog, and lets you perform a query. It returns modal result "OK" and, if the user has performed a query, it returns a ViewWindow object containing the query results.

The query results are extracted from the ViewWindow object through its associated View object. To select specific objects from the query results you would need to display the query results on a separate View.

Note that the ExecuteSelectFromQueryResult Dialog allows you to select from the query results in the same dialog as you executed the query.

Example

This example runs the ExecuteQueryByAttributes method and extracts the query results to a ComponentObjects object.

```
Dim GUI As SmGUISrv.SmCommonGUI

Dim SmView As SmGUISrv.ISmView ' view properties and data
```

```
Dim SmViewWindow As SmUISrv.ISmViewWindow ' window methods

Dim ComponentObjects As ISmComponentObjects

Set SmViewWindow = Nothing

GUI.Dialogs.ExecuteQueryByAttributes SmViewWindow

Set ComponentObjects = Nothing

If Not SmViewWindow Is Nothing Then

    Set SmView = SmViewWindow.SmView

    If Not SmView Is Nothing Then

        Set ComponentObjects = SmView.DisplayObjects

    End If

    SmViewWindow.Close

End If

If Not ComponentObjects Is Nothing Then

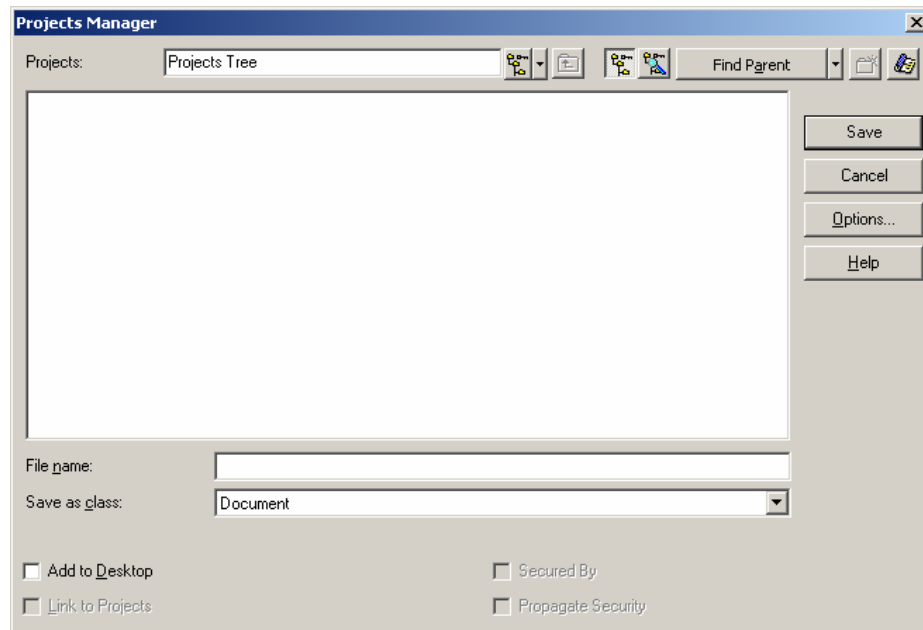
    MsgBox ComponentObjects.CompositeObjects.Count

    MsgBox "Not nothing"

End If
```

ISmSaveAsDialog.ControlProperties

The ISmSaveAsDialog.ControlProperties is a property of the ISmSaveAsDialog Object which influences the behavior of the Save As Dialog, for example:



The `ISmSaveAsDialog.ControlProperties` is a collection of Properties which influence the appearance of the Save As dialog box.

These collections of Properties are of the `ISmGUIProperties` Type.

Those Properties support attributes of a screen component, such as *visible*, *enabled* ... The attributes of those Properties are members of the `ISmGUIProperty`.

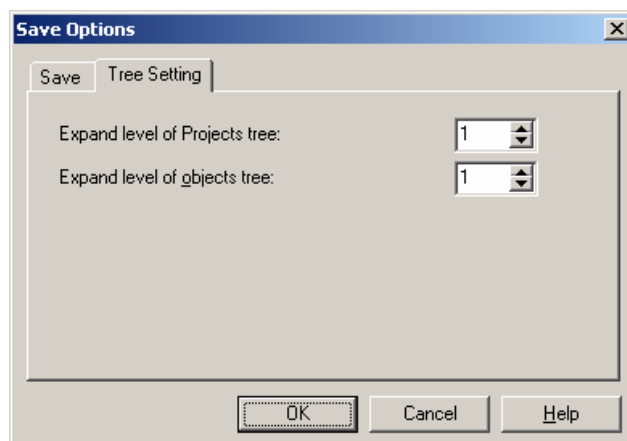
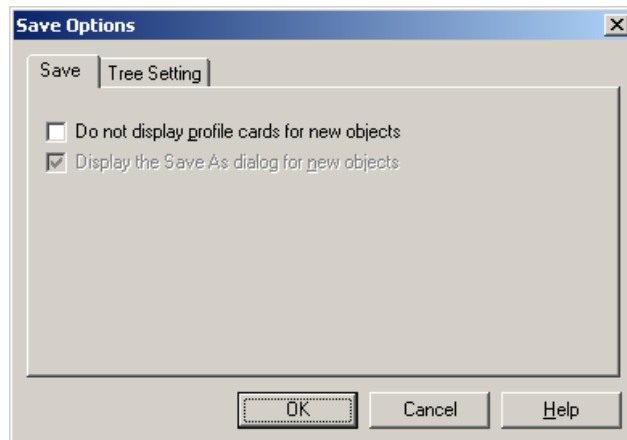
Every Property is recognized by its associated Screen Control - via the Name attribute of the property.

The following Property Names are supported and possibly appear in the collection:

Method	Description
frmSaveAsDialog	Enables the Caption of the Save As window
edtFileName	Enables the Visible, Enabled, and Read-Only attributes, in the " <i>File Name</i> " edit control
lblFileName	Enables the Visible, Enabled and Caption in the "File Name" label
cbxSaveAsClass	Enables the Visible and Enabled attributes, in the "Save As Class" name box
lblSaveAsClass	Enables the Visible, Enabled and Caption in the "Save As Class" label
chkDontDisplayAgain	Enables the Visible, Enabled, Checked and Caption in the "Do not display the Save As dialog" checkbox
chkAddToDesktop	Enables the Visible, Enabled, Checked and Caption in the "Add To Desktop" checkbox
chkLinkToMainClass	Enables the Visible, Enabled, Checked and Caption in the "Link To Class" checkbox
chkSecuredBy	Enables the Visible, Enabled, Checked and Caption in the "Secured By" checkbox
chkPropagateSecurity	Enables the Visible, Enabled, Checked and Caption in the "Propagate Security" checkbox
clsClassSettings	Enables the Class Settings through ISmRecordList. The RecordList is transferred by the Value attribute in the Property

ISmSaveAsDialog.OptionsProperties

The ISmSaveAsDialog.OptionsProperties is a property of the ISmSaveAsDialog Object that influences the behavior of the Options dialog box (when the options button is clicked), for example the following dialog boxes:



ISmSaveAsDialog.OptionsProperties is a collection of Properties that influence the appearance of the Save As dialog box.

These options can be altered later by clicking the Options button on the Save As Dialog, and changing the values in the Save Options dialog box.

These Option Properties are of the ISmGUIProperties Type.

The Option Properties support attributes in the Save Options dialog box of screen component, for example, *visible*, *enabled* ...

Note: The attributes are members of ISmGUIProperty.

The Option is recognized by its associated Screen Control - via the Name attribute of the option.

The following Option Names are supported and possibly appear in the collection:

Method	Description
chkBatchMode	Enables the Visible, Enabled, Checked, Caption in the "Do not display profile cards for new objects" checkbox
chkDisplayDialogForNewObject	Enables the Visible, Enabled, Checked, Caption in the "Display the Save As" dialog in the new objects' checkbox
lblMainClassExpandLevelTree	Enables the Visible, Enabled and Caption of the Expand level of ... tree label
spnedtMainClassExpandLevelTree	Enables the Visible, Enabled, Value of the "Expand level of ... tree" in the Spin Edit Control
lblObjectsExpandLevelTree	Enables the Visible, Enabled, Value of the "Expand level of ... tree" in the Spin Edit Control
spnedtObjectsExpandLevelTree	Enables both, the Visible, Enabled, Value of the "Expand level of ... tree" Spin Edit Control, and the Visible, Enabled, Value of the "Expand level of objects tree" Spin Edit Control

ISmLocalFilesExplorer

The ISmLocalFilesExplorer object represents the **SmarTeam** Local Files Explorer dialog.

The object diagram of ISmLocalFilesExplorer is shown below:

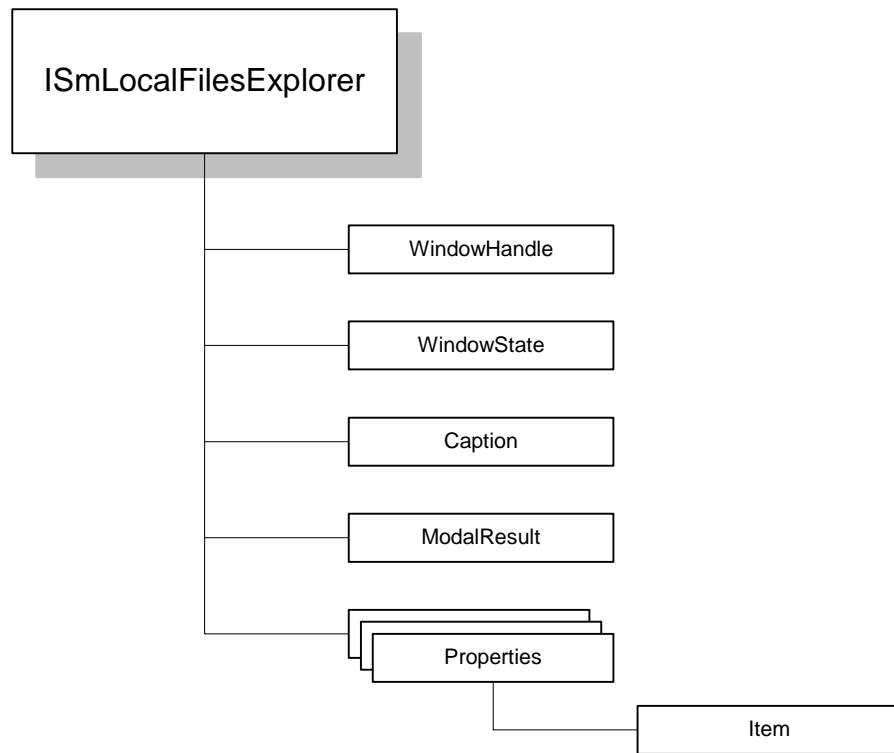


Figure 6-9 *ISmLocalFilesExplorer* Object Diagram

Properties

The ISmLocalFilesExplorer object has the following properties and sub-properties:

Property	Description
WindowHandle	Handle from application that called the Local Files Explorer
WindowState	The current display state of the Local Files Explorer window <ul style="list-style-type: none">• wstMaximized• wstMinimized• wstNormal
Caption	Sets caption
ModalResult	Returns modal result
Properties	Returns ISmWindowProperties. See below for details.

Methods

The ISmLocalFilesExplorer object has the following methods:

Method	Description
Show	Shows the window
Hide	Hides the window
Close	Closes the window
BringToFront	Brings window to front
ShowModal	Shows window in modal mode
Refresh	Rebuild current directory in Local Files Explorer. This method must be called for each change in the life cycle, for example when adding or deleting an object.

Example

```
Dim FSmLocalFilesExplorer as new ISmLocalFilesExplorer
` CommonGUI as SmCommonGUI
` Open Local Files Explorer

FSmLocalFilesExplorer = CommonGUI.Dialogs.NewLocalFilesExplorer
` Refresh the Local Files Explorer

FSmLocalFilesExplorer.Refresh
```

ISmWindowProperties

The ISmWindowProperties object is a collection of properties associated with the window.

Properties

The ISmWindowProperties object has the following properties and sub-properties:

Property	Description
Item	Individual window property.

Methods

The ISmWindowProperties object has the following methods:

Method	Description
Clear	Clears a property

ISmSaveAsDialog

The ISmSaveAsDialog object represents the SaveAs dialog.

The object diagram of ISmSaveAsDialog is shown below:

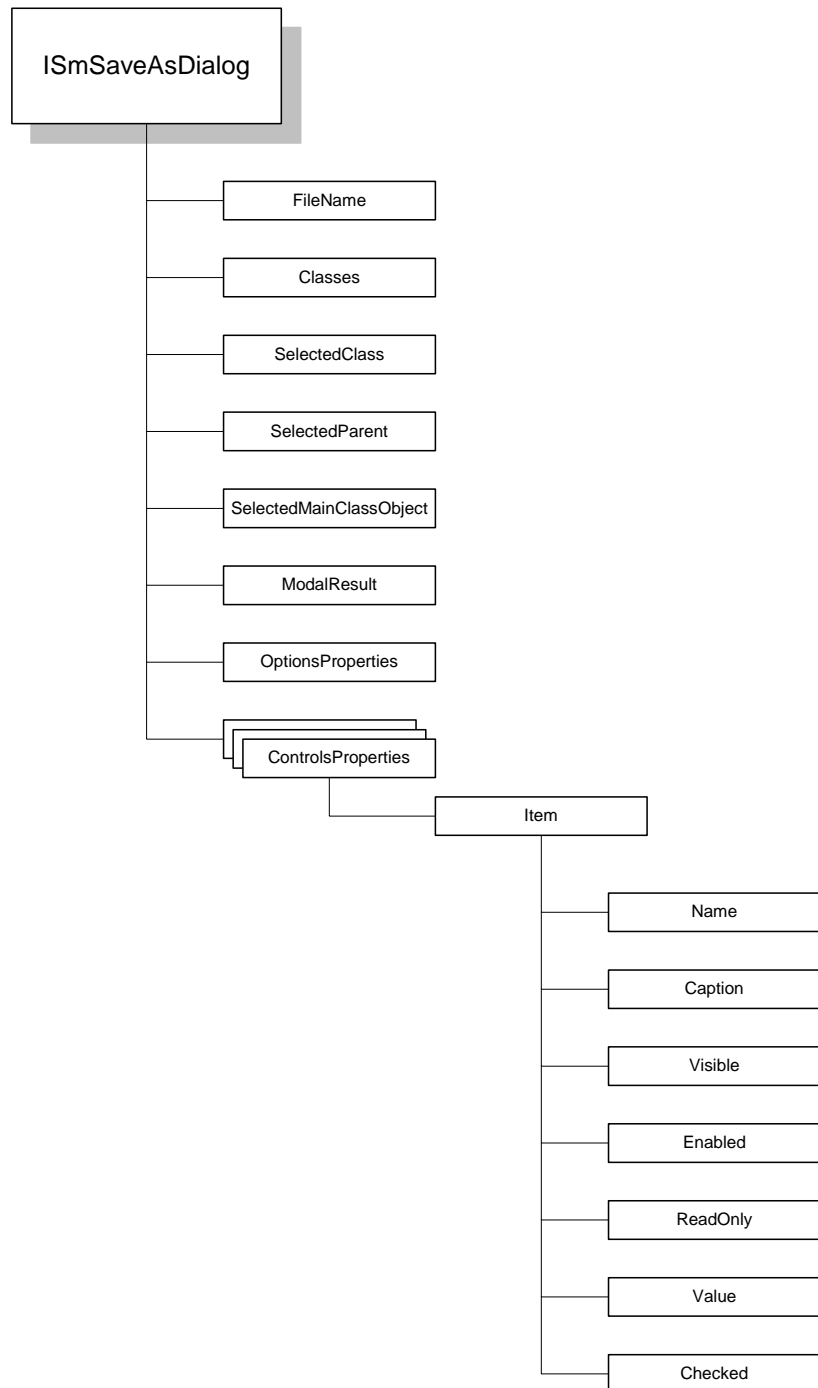


Figure 6-10 *ISmSaveAsDialog* Object Diagram

Properties

The ISmSaveAsDialog object has the following properties and sub-properties:

Property	Description
FileName	Returns and sets name of the file.
Classes	Returns and sets SmClasses object representing list of classes that appear in the "Save as Class" drop-down list.
SelectedClass	Returns or sets SmClass object representing selected class.
SelectedParent	Returns or sets SmObject object representing selected parent
SelectedMainClassObject	Returns or sets SmObject object representing the selected project.
ModalResult	Modal result of the window
OptionsProperties	Returns or sets option properties. Returns ISmGUIProperties. See below for details.
ControlsProperties	Returns or sets control properties. Returns ISmGUIProperties

Methods

The ISmSaveAsDialog object has the following methods:

Method	Description
ShowModal	Displays window in modal mode.

ISmGUIProperties

The ISmGUIProperties object is a collection of ISmGUIProperty objects.

Properties

The ISmGUIProperties object has the following properties:

Property	Description
Item	Returns ISmGUIProperty.

Methods

The ISmGUIProperties object has the following methods:

Method	Description
NewProperty	Adds new SmGUIProperty object to collection specified by name. Returns ISmGUIProperty
ItemByName	Returns a member of a collection by its name. Returns ISmGUIProperty
Remove	Removes SmGUIProperty object from collection.
Clear	Clears all SmGUIProperty objects from collection.

ISmGUIProperty

The ISmGUIProperty object represents the visual settings of the SaveAs dialog.

Properties

The ISmGUIProperty object has the following properties:

Property	Description
Name	Returns or sets name of the property.
Caption	Returns or sets caption of the property.
Visible	True if property is visible.
Enabled	True if property is enabled.
ReadOnly	True if property is read only.
Value	Returns or sets value of the property.
Checked	True if property is set (checked).

Example

The following example displays the Save As dialog.

```
Private Sub DisplaySaveAsDialog(SmSession as ISmSession, FileName As String,  
ProjectObject As ISmObject)
```

```
Begin
```

```
    Set ClassesList = SmSession.MetaInfo.NewSmClasses
```

```
    ClassesList.Add(354)
```

```
    ClassesList.Add(353)
```

```
    Dim SmGUIServices As ISmGUIServices
```

```
    Set SmGUIServices = SmSession.GetService("SmGUISrv.SmGUIServices")
```

```
    Set GUIStore = SmGuiServices.GUIStore
```

```
    Set SaveAsDialog = GUIStore.NewSaveAsDialog
```

```
    ' Set input parameter
```

```
    SaveAsDialog.Classes=ClassesList
```

```
    SelectedMainClassObject =
```

```
SmSession.ObjectStore.NewObject(ProjectObject.ClassId)
```

```
    SelectedMainClassObject.ObjectId = ProjectObject.ObjectId
```

```
SaveAsDialog.SelectedMainClassObject = SelectedMainClassObject  
  
SaveAsDialog.FileName= FileName  
  
SaveAsDialog.ShowModal  
  
End Sub
```

ISmOpenDialog

The ISmOpenDialog object represents the Open dialog.

The Open dialog has the look and feel of Office 2000. An outlook bar on the left side of the dialog provides different ways of locating a specific document.

This dialog has three functions:

1. Search – Running a pre-defined ‘parametric’ query and selecting one or more document from the results. You can define a wide range of parameters and parameter criteria.
2. Smart Desktop – Browsing through project data.
3. System Folders – Returns to called function with specific return code

Search

The Search function of the OpenFileDialog is shown below:

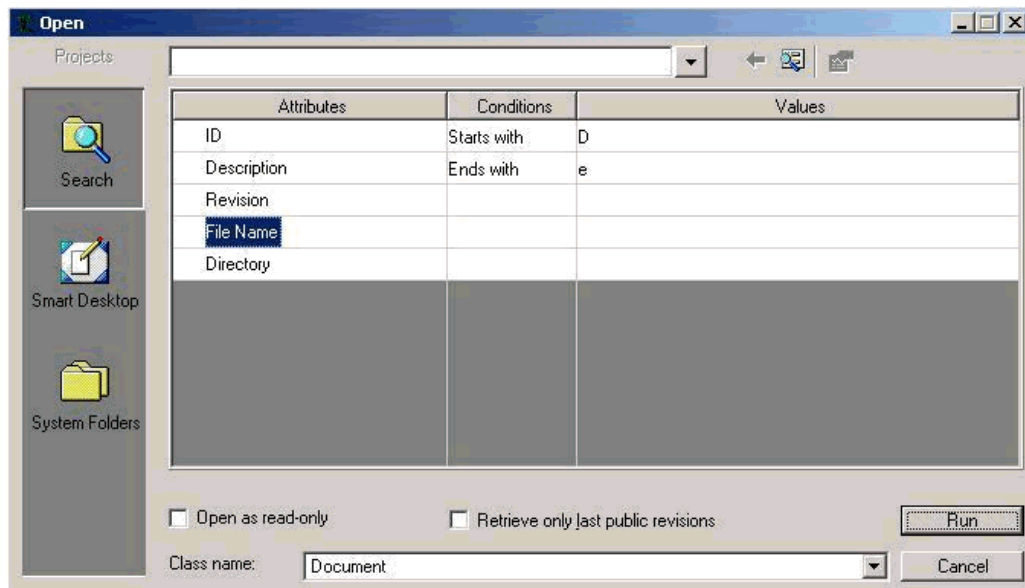


Figure 6-11 ISmOpenDialog Search Function

Smart Desktop

The Open Dialog Smart Desktop is shown below.

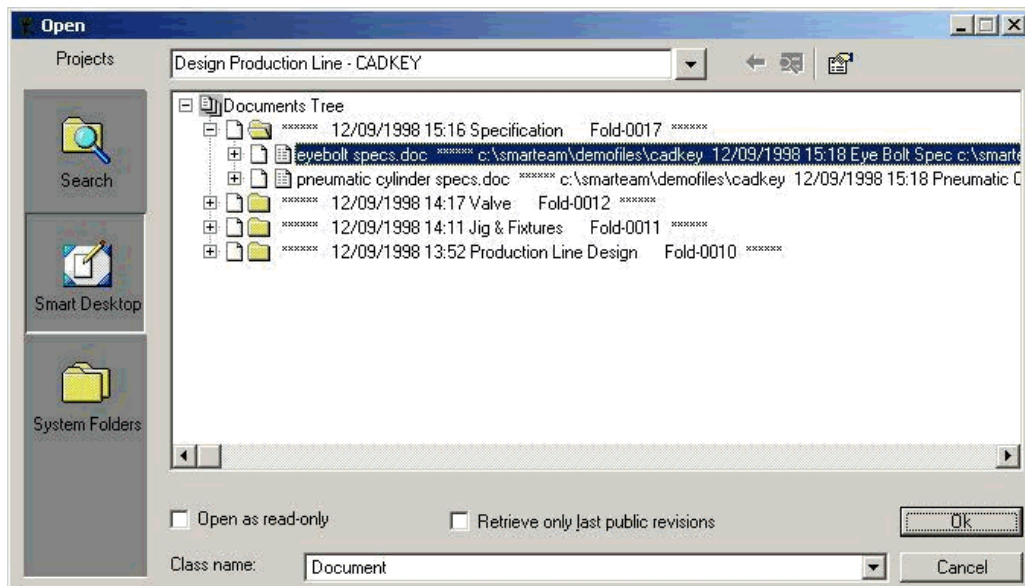


Figure 6-12 ISmOpenDialog Smart Desktop Function

Object Diagram

The object diagram of ISmOpenDialog is shown below:

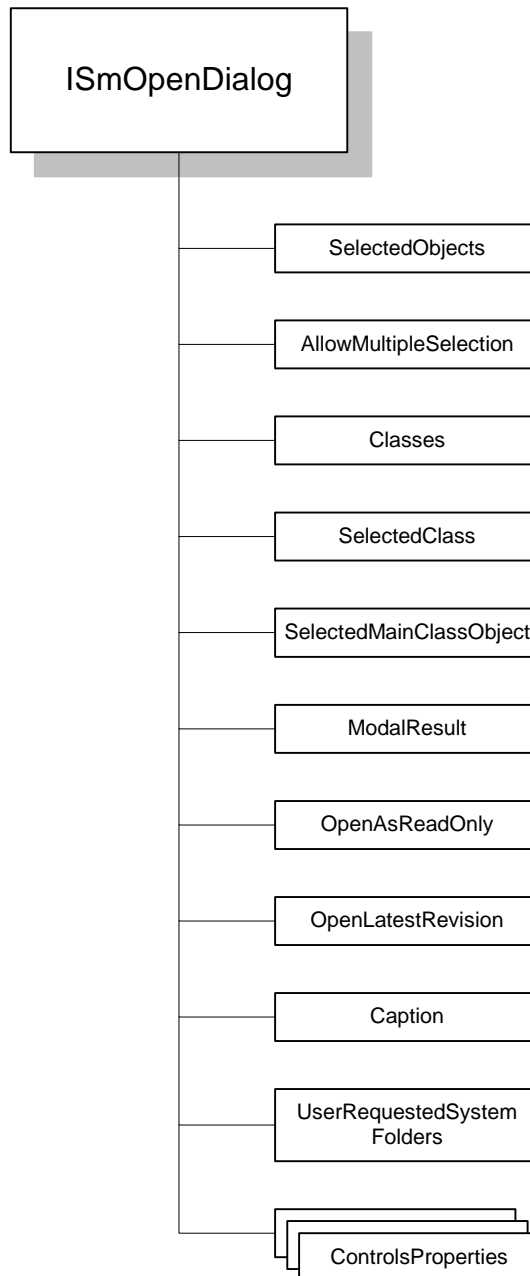


Figure 6-13 ISmOpenDialog Object Diagram

Properties

The ISmOpenDialog object has the following properties:

Property	Description
SelectedObjects	Collection of user-selected objects
AllowMultipleSelection	Allows user to select more than one object from window
Classes	A Collection of classes from which user can select object (can be SuperClasses)
SelectedClass	Sets and returns a user-selected class (set for default).
SelectedMainClassObject	Sets and returns user-selected main object (set also for default – chosen on show)
ModalResult	Modal result for dialog – Can be either mrOk or mrCancel for the System Folders button can be mrRetry
OpenAsReadOnly	Boolean. Used to receive additional information from user
OpenLatestRevision	Boolean. Used to receive additional information from user true, in case the selected objects are revision-managed the latest revisions are opened.
Caption	View caption
UserRequestedSystemFolders	Boolean. If true, shows the System Folders button on the View
ControlsProperties	Returns ISmGUIProperties (see section ISmGUIProperties)

Methods

The ISmOpenDialog object has the following methods:

Method	Description
ShowModal	Displays window in modal mode.

Example

The following example shows how to use the OpenDialog object.

```
Sub OpenDialogTest(GUI As SmGUISrv.SmCommonGUI)

    Dim OpenDialog As SmGUISrv.ISmOpenDialog

    Set OpenDialog = GUI.Dialogs.NewOpenDialog

    Set OpenDialog.Classes = Session.MetaInfo.NewSmClasses

    OpenDialog.Classes.AddClass
    Session.MetaInfo.SmClassByName("Document").ClassId

    OpenDialog.ShowModal

    If OpenDialog.ModalResult = mrOK then

        MsgBox "User selected " + Cstr(OpenDialog.SelectedObjects.Count) +
        " objects"

    Else

        MsgBox "User has not selected any objects"

    End if

End Sub
```

7. SmarTeam Utilities Library

General Description

The **SmarTeam** Utilities library provides the following functionality:

- Format conversions
- Mask creation and attribute definition
- Life cycle functionality
- Other miscellaneous functionality.

Dependencies

The **SmarTeam** Utilities library has the following dependencies:

- **SmarTeam** Record List library
- **SmarTeam** Engine library.

Overview of Objects

The object hierarchy of the **SmarTeam** utility library objects is shown below:

Library scheme

SmUtil

 SmSessionUtil

 SmSessionConvert

 SmMiscUtil

 SmMiscUtil

 SmConvert

SmSessionUtil Object

The **SmSessionUtil** object provides various session-related utility functions, such as options to specify preferences for the current session, and to retrieve objects by their CAD ID.

SmSessionUtil is a service object. To obtain a reference to the object, use the following syntax:

```
Dim SessionUtil as SmSessionUtil

Set SessionUtil = SmSession.GetService("SmSessionUtil")
```

Object scheme

SmSessionUtil

 SmSessionConvert (SmSessionConvert object)

 SmMiscUtil (SmMiscUtil object)

Register

CheckIn

CurrentSessionPreference

CopyFileFromVault

...

Object Functionality

The object has the following functionality:

- File vault operations
- Copied-file registration
- Life-cycle operations
- Life-cycle authorization operations
- Mask operations
- Miscellaneous utilities.

File Vault Operations

This section describes API functions that copy objects and files in and out of vaults. Vaults are secure directories where **SmarTeam** stores an object document in the different life-cycle stages of the object. A vault can represent a directory in a remote computer. Thus the vault identifier is required to uniquely define the file location, since the same directory name could exist in two different vault computers.

A vault is represented by an `SmObject`. There is no persistent object (for example, `SmVault`) that represents a vault. Instead, a `SmObject` is defined with the necessary properties of the vault. The object property `Vault.ObjectId` is the unique vault identifier.

Vault objects are active when the vault server is active. When working locally, vault objects are ignored and the `nil` Object Id replaces the `Vault.ObjectId` as the vault function parameter `SourceVault`.

Three types of vaults are defined where each type of vault corresponds to a life-cycle stage. The vault type is represented by `VaultTypeEnum`.

Each object is associated with one or more vaults for each life-cycle stage, where one of them is assigned to be the default vault for the object for that life-cycle stage.

The following types of vaults exist:

Vault Type (VaultTypeEnum)	Description
VltApprove	Vault for released documents
VltInWork	Vault for checked in documents
VltObsolete	Vault for obsolete documents

The SmSessionUtil object contains the following functions to support file vault operations.

Function	Description
CompareFile	Compares files of two objects
CopyFileExtended	Copy file between two vaults
CopyFileFromVault	Copies file out of vault
CopyObjectFileFromVaultPermission	Copies file of object out of vault and sets file access permission.
CopyObjectFileToVault	Copies file of object to vault
DeleteFileFromVault	Deletes a file from a vault
FileExists	Checks if file exists in vault
GetPossibleVaultsForObject	Gets all vaults associated with object
GetVaultDirectoryForObject	Gets directory of object in vault
GetVaultForObject	Gets default vault associated with object
MoveFileToVault	Moves file to vault

File Vault Task: Copying a File from a Vault

You can use one of the following functions to copy a file from a vault

Function	Description
CopyFileFromVault	To copy a file from a vault to an external directory when you know the file name, file location and vault
CopyFileExtended	To copy a file from one vault to another when you know the file names, file locations and vault ids.
CopyObjectFileFromVaultPermission	To copy a file from a vault to an external directory when you know the object that contains the file and destination file name and directory. This function also sets file read/write access permission for the copied file.

InsteadOf CopyFile

This script is used in the InsteadOf hook of the CopyFile SmarTeam operation. It can copy a file to the work directory and determine the read/write access permission.

```
Function InsteadOfCopyFileExample(ApplHndl As Long,Sstr As String,FirstPar As Long,SecondPar As Long,ThirdPar As Long ) As Integer
```

```
    Dim SmSession As SmApplic.SmSession
```

```
    Dim FirstRec As Object
```

```
    Dim SecondRec As Object
```



```
Dim ThirdRec As Object

Dim FirstWorkObject As SmApplic.ISmObject

Dim SecondWorkObject As SmApplic.ISmObject

Dim Vault As SmApplic.ISmObject ' vault as object

'Dim Vault As Object ' vault as object

Dim VaultType As Integer ' type of vault

Dim SourceVault As Long ' source vault object id

Dim SourceDirectory As String 'source directory in vault

Dim SourceFileName As String 'source file name

Dim DestinationDirectory As String 'destination directory in vault

Dim DestinationFileName As String 'destination file name

Dim SessionUtil As SmUtil.SmSessionUtil 'main SmarTeam service object

Dim Result As Long ' object id for refresh

Dim FileMode As Integer 'file mode in destination directory

' Convert pointer to COM object SmSession
Set SmSession = SCREXT_ObjectForInterface(ApplHndl)

' Conver input parameter to COM object
CONV_RecListToComRecordList FirstPar,FirstRec
CONV_RecListToComRecordList SecondPar,SecondRec
CONV_RecListToComRecordList ThirdPar,ThirdRec

Set FirstWorkObject =
SmSession.ObjectStore.ObjectFromData(FirstRec.GetRecord(0),true)

Set SecondWorkObject =
SmSession.ObjectStore.ObjectFromData(SecondRec.GetRecord(0),true)

' Get service object
Set SessionUtil = SmSession.GetService("SmUtil.SmSessionUtil")

' Vault type
```

```
VaultType = VltInWork

' Get Vault for WorkObject according to specific vault type
Set Vault = SessionUtil.GetVaultForObject(VaultType, FirstWorkObject)

' Get Vault object id - for local vault NULL object id
If Vault Is Nothing Then

    SourceVault = NULL_OBJ_ID

Else

    SourceVault = Vault.ObjectId

End If

' Set all parameters
SourceDirectory = FirstWorkObject.Data.ValueAsString(NM_DIRECTORY)
DestinationDirectory = SecondWorkObject.Data.ValueAsString(NM_DIRECTORY)

SourceFileName = FirstWorkObject.Data.ValueAsString(NM_FILE_NAME)
DestinationFileName = SecondWorkObject.Data.ValueAsString(NM_FILE_NAME)

FileMode = modReadWrite 'file mode in destination directory

' Perform operation on object using SmSessionUtil method
'CopyFileFromVault

'Result = SessionUtil.CopyFileFromVault(SourceVault, SourceDirectory,
SourceFileName, DestinationDirectory, DestinationFileName)

'CopyObjectFileFromVaultPermission

SessionUtil.CopyObjectFileFromVaultPermission FirstWorkObject,
DestinationFileName, DestinationDirectory, FileMode

InsteadOfCopyFileExample = Err_None

End Function
```

Copied-File Registration

This section describes API functions that manage copied-file registration. When a file associated with an object is copied to a different directory using the **SmarTeam** commands Check Out, New Release, Copy File or View, the copied file is registered. For example, executing Copy File on a Document copies the associated file to the work directory and registers the copied file. The Local File Explorer GUI lets you keep track of all registered copied files and lets you conveniently delete them when they are not needed.

The following table shows the four possible status values of a file in the copied-file registration:

Status of Copied-File in Copie File Registration	Default Copied-File Location	SmarTeam command used to co file
Checked out	/work	Check Out, New Release
Copied file	/work	Copy File
Copied and referenced file	/work	Copy File
Viewed file	/view	View

The default locations mentioned in the table are specified in the System Configuration Editor under “Miscellaneous Configuration/Directory Structure”. The key “USER_DIR” is the default location for the Check Out, Copy File and New Release operations. The key “ReadOnlyDir” is the default location for the View operation.

The SmSessionUtil object contains the following functions to support copied-file registration.

Function	Description
AddReferenceToFileCopy	Adds a reference to copy file maintenance (TDM_COPY_FILE table).
DeleteAllFilesRegistration	Deletes all registered files
DeleteCopiedFilesRegistration	Deletes only registered copied (or copied and referenced) files
DeleteFilesRegistrationForObject	Deletes all files registered for an object
DeleteMissingFilesRegistration	Deletes registration records when the corresponding files are not found
DeleteSpecificFileRegistrationForObject	Deletes one of the files registered for an object
DeleteViewedFilesRegistration	Deletes all registered viewed files

Common Tasks

Copied-File Registration Task: Delete a Viewed File for an Object

The following script uses `DeleteSpecificFileRegistrationForObject` to delete a viewed file for an object, including the file and the registration record. It is assigned as a user-defined function, which is executed when the object is selected on the **SmarTeam** view.

```
Function DeleteViewedFileRegistrationForObject(ApplHndl As Long,Sstr As
String,FirstPar As Long,SecondPar As Long,ThirdPar As Long ) As Integer

    Dim SmSession As SmApplic.SmSession

    Dim FirstRec As Object

    Dim SecondRec As Object

    Dim ThirdRec As Object

    Dim SessionUtil    As SmUtil.SmSessionUtil 'main SmarTeam service object

    Dim WorkObject As SmApplic.ISmObject

    Dim FileName As String 'file name for object

    Dim ViewDirectory As String 'directory

    Set SmSession = SCREXT_ObjectForInterface(ApplHndl)

    Set SessionUtil = SmSession.GetService("SmUtil.SmSessionUtil")

    ' Conver input parameter to COM object

    CONV_RecListToComRecordList FirstPar,FirstRec

    Set WorkObject =
SmSession.ObjectStore.ObjectFromData(FirstRec.GetRecord(0),true)

    WorkObject.Retrieve 'to get file name

    ViewDirectory= "E:\Program Files\View"

    FileName = WorkObject.Data.ValueAsString(NM_CAD_REF_FILE_NAME) 'original
file name, same as copied file name

    MsgBox "Deleting viewed file " + ViewDirectory + " " + FileName + " for
object " + Str(WorkObject.ObjectId)
```

```

'deletes file and appearance in local file explorer

SessionUtil.DeleteSpecificFileRegistrationForObject WorkObject, FileName,
ViewDirectory

DeleteViewedFileRegistrationForObject = Err_None

End Function

```

Lifecycle Operations

The SmSessionUtil object contains various types of life-cycle functionality for persistent objects.

The life cycle functionality is divided into two groups:

- Individual Operations
- Group Operations

Individual Operations

These methods perform life-cycle operations on individual objects. The methods check for user authorization and whether the object is in a valid state for the operation. If the object is not valid, an error message is displayed. The table indicates if the method can call scripts.

Function	Description	Calls scripts: <u>Optionally</u> <u>Always</u> <u>Never</u>
Approve	Release the object	O
CheckIn	Check in the object	O
CheckOut	Check out the object	O
NewRelease	New-release the object	O
Obsolete	Obsolete the object	O
Register	Register the object	O
UndoCheckOut	Undo the previous checkout operation. Always calls script	A
UndoCheckOutEx	Undo the previous checkout operation. Can set behavior preference to not call scripts.	O
HandleCheckInApprove	Check in or Release object Objects are not locked.	N
HandleCheckOutNewRelease	Check out or New-release ob	N
HandleRegisterFreeze	Objects are not locked. Register or Obsolete object	N

Objects are not locked.

Operations for Part Objects

The following operations are provided for Part objects, which are subject to Part Class Behavior.

Function	Description
Promote (operation)	Promotes the state of the Part object. The operation is executed using the method ExecuteOperationOnObjectTree as described below.
NewPartRevision	Returns a new Part Revision for the specified SourceObject and attribute values.

Task Record Argument for Individual Life-Cycle Operations

The individual life-cycle operation methods have the following typical format:

```
Function CheckIn(  
    SourceObject As ISmObject,  
    TaskRecord As ISmRecord,  
    InvokeScripts As Boolean  
) As Integer
```

Most of the individual life-cycle methods require the ISmRecord argument TaskRecord – an object that represents attributes for the life-cycle operation being performed.

- If you want the life-cycle operation to be performed with the default attribute values, set the TaskRecord to null.
- If you want to specify attribute values for the life-cycle operation, which are different from the default values, you need to load them into the TaskRecord object. Figure 14 shows the structure of the TaskRecord object. For each attribute, you load its header and its value.

Figure 14 Task Record for a Life-Cycle Operation

Life-Cycle Operation Attributes				
Header: Name	Attribute-1	Attribute-2	...	Attribute-n
Type	Type-1	Type-2		Type-n
Size	Size-1	Size-2		Size-n
LC Operation	Value-1	Value-2	...	Value-n

Example

For example, the following creates a Task Record for the Check In operation and specifies the attributes: NM_LFCYC_CHECKIN_MODE and NM_REVISION. This code operates in the InsteadOf CheckIn hook. The attribute values are received from **SmarTeam** in SecondRec and loaded into the TaskRecord.

```
' Create task record

Set TaskRecord=CreateObject("SmRecList.SmRecord")

' Add task to task record

TaskRecord.AddHeader NM_LFCYC_CHECKIN_MODE, 2, sdtSmallInt

TaskRecord.ValueAsSmallInt(NM_LFCYC_CHECKIN_MODE) =
SecondRec.ValueAsSmallInt(NM_LFCYC_CHECKIN_MODE, 0)

TaskRecord.AddHeader NM_REVISION, 256, sdtChar

TaskRecord.ValueAsString(NM_REVISION) =
SecondRec.ValueAsString(NM_REVISION, 0)
```

Individual Life-Cycle Operation Task: Check In an Individual Object

The following script uses the Register and the Check In function to check an object into the vault depending on its state. It is defined as a script hook function in the InsteadOf hook of the Check In operation. Check In does all authorization checks.

```
Function InsteadOfCheckInExample(ApplHndl As Long,Sstr As String,FirstPar As
Long,SecondPar As Long,ThirdPar As Long ) As Integer

Dim SmSession As SmApplic.SmSession

Dim FirstRec As Object

Dim SecondRec As Object

Dim ThirdRec As Object

Dim Operation As SmApplic.ISmOperation 'performed operation object

Dim Metainfo As SmApplic.SmMetaInfo 'metainfo object for smClasses

Dim SessionUtil As SmUtil.SmSessionUtil 'main SmarTeam service object

Dim TaskRecord As Object 'task record for operation
```

```
Dim OperName As String 'operation name

Dim InvokeScripts As Boolean 'invoke scripts on operation

Dim Result As Long 'result of operation

Dim CheckinMode As Integer

Dim WorkObject As SmApplic.ISmObject

Dim State As Integer

Dim NewLookupObj As SmApplic.ISmLookupObject

Dim CheckedOutLookupObj As SmApplic.ISmLookupObject

Dim StateClass As SmApplic.ISmClass


Set SmSession = SCREXT_ObjectForInterface(ApplHndl)

Set Metainfo = SmSession.Metainfo

' Convert pointer to COM object SmSession

CONV_RecListToComRecordList FirstPar,FirstRec

CONV_RecListToComRecordList SecondPar,SecondRec

CONV_RecListToComRecordList ThirdPar,ThirdRec

' Get service object

Set SessionUtil = SmSession.GetService("SmUtil.SmSessionUtil")

' Invoke scripts on operation execution (before, after, instead)

InvokeScripts = False

' Conver input parameter to COM object

Set WorkObject =
SmSession.ObjectStore.ObjectFromData(FirstRec.GetRecord(0),true)

' Create task record

Set TaskRecord=CreateObject("SmRecList.SmRecord")

' Add task to task record

TaskRecord.AddHeader NM_LFCYC_CHECKIN_MODE, 2, sdtSmallInt

TaskRecord.ValueAsSmallInt(NM_LFCYC_CHECKIN_MODE) =
SecondRec.ValueAsSmallInt(NM_LFCYC_CHECKIN_MODE, 0)
```



```
TaskRecord.AddHeader NM_REVISION, 256, sdtChar

'Determine object STATE to choose between REGISTER and CHECKIN

State = FirstRec.ValueAsInteger(NM_STATE, 0)

Set StateClass = SmSession.MetaInfo.GetInternalSmClass("TDM_STATE")

Set NewLookupObj =
SmSession.ObjectStore.GetSmLookUpByUniqueName(StateClass.ClassId, "New")

Set CheckedOutLookupObj =
SmSession.ObjectStore.GetSmLookUpByUniqueName(StateClass.ClassId, "Checked
Out")

If Not (WorkObject Is Nothing) Then

    If State = NewLookupObj.Id Then    'new - register

        ' new revision

        TaskRecord.ValueAsString(NM_REVISION) = "x"

        ' Get operation object

        OperName = NM_OPER_REGISTRATION

        Set Operation = MetaInfo.OperationsForClass(WorkObject.ClassId,
False).ItemByName(OperName)

        ' Perform CheckIn operation on object using SmSessionUtil method

        Result = SessionUtil.Register(WorkObject, TaskRecord,
InvokeScripts)

    ElseIf State = CheckedOutLookupObj.Id Then    'checked out

        ' Set operation name according to default constant

        OperName = NM_OPER_CHECKIN

        Set Operation = MetaInfo.OperationsForClass(WorkObject.ClassId,
False).ItemByName(OperName)

        TaskRecord.ValueAsString(NM_REVISION) =
SecondRec.ValueAsString(NM_REVISION, 0)

        Result = SessionUtil.CheckIn(WorkObject, TaskRecord, InvokeScripts)

    End If

    MsgBox "Deleted Object ID " + Str(Result)
```

```
End If

InsteadOfCheckInExample = Err_None

End Function
```

Individual Life-Cycle Operation Task: Check-in an Individual Object with the NM_LFCYC_CHECKIN_MODE task

The following script is hooked to 'Before Check in' to handle the file name change when changing task NM_LFCYC_CHECKIN_MODE.

```
If Sstr <> NM_OPER_CHECKIN Then Exit Function

Dim SmObj As ISmObject, PrevObj As ISmObject

Set SmObj = Session.ObjectStore.ObjectFromData(RecList1.GetRecord(0),
True)

Dim Util As SmSessionUtil

Set Util = Session.GetService("SmUtil.SmSessionUtil")

Set PrevObj = Util.GetObjectByRevision(SmObj, SmObj.Data.Value(NM_PAR_
REVISION))

If SmObj.ObjectId = PrevObj.ObjectId Then Exit Function ' if no previous
revision exists

Dim filename As String, name As String, ext As String, pos As Integer

filename = SmObj.Data.Value(NM_FILE_NAME)

pos = InStr(filename, ".")

name = Left(filename, pos - 1)

ext = Right(filename, Len(filename) - pos)

If RecList2.Value(NM_LFCYC_CHECKIN_MODE, 0) = LFCYC_WorkRev Then

    RecList2.Value(NM_LFCYC_CHECKIN_MODE, 0) = LFCYC_PrevRev ' In 'Replace
previous revision' we have to provide the FILE_NAME task of previus
object

RecList2.Value(NM_FILE_NAME, 0) = PrevObj.Data.Value(NM_FILE_NAME)

Else
```

```
RecList2.Value(NM_LFCYC_CHECKIN_MODE, 0) = LFCYC_WorkRev ' ' In 'Current  
revision' we have to build the FILE_NAME task of current object
```

```
RecList2.Value(NM_FILE_NAME, 0) = name & "_" & SmObj.ObjectId & "_" &  
SmObj.ClassId & "." & ext
```

```
End If
```

Individual Lifecycle Task: Check Out an Individual Object

The following script uses the HandleCheckOutNewRelease function to check out an object from the vault. It is defined as a script hook function in the InsteadOf hook of the Check Out operation.

```
Function HandleCheckOut(ApplHndl As Long,Sstr As String,FirstPar As  
Long,SecondPar As Long,ThirdPar As Long ) As Integer  
  
    Dim SmSession As SmApplic.SmSession  
  
    Dim FirstRec As Object  
  
    Dim SecondRec As Object  
  
    Dim ThirdRec As Object  
  
    Dim Operation As SmApplic.ISmOperation 'performed operation object  
  
    Dim NewWorkObject As SmApplic.ISmObject 'new object  
  
    Dim WorkObject As SmApplic.ISmObject 'new object  
  
    Dim MetaInfo As SmApplic.SmMetaInfo 'metaInfo object for smClasses  
  
    Dim SessionUtil As SmUtil.SmSessionUtil 'main SmartTeam service object  
  
    Dim TaskRecord As Object 'tasks record for operation - for SmartScript  
  
    Dim OperName As String 'operation name  
  
    Dim Result As Long 'result of operation - new object id for refresh  
  
    Dim WorkDir As String 'work directory  
  
    Dim RetValue As Integer  
  
    Dim TreatCommonFileObjects As Boolean  
  
    ' Convert pointer to COM object SmSession
```

```
Set SmSession = SCREXT_ObjectForInterface(ApplHndl)

' Conver input parameter to COM object
CONV_RecListToComRecordList FirstPar,FirstRec
CONV_RecListToComRecordList SecondPar,SecondRec
CONV_RecListToComRecordList ThirdPar,ThirdRec

' Get service object
Set SessionUtil = SmSession.GetService("SmUtil.SmSessionUtil")

' Conver input parameter to COM object
Set Metainfo = SmSession.Metainfo

Set WorkObject =
SmSession.ObjectStore.ObjectFromData(FirstRec.GetRecord(0),true)

' Create task record
Set TaskRecord=CreateObject("SmRecList.SmRecord")

' Add task to task record
TaskRecord.AddHeader NM_FILE_NAME, 255, sdtChar

' Add task to task record
TaskRecord.AddHeader NM_DIRECTORY, 255, sdtChar

' Set destination file name for task record
TaskRecord.ValueAsString(NM_FILE_NAME) =
SecondRec.ValueAsString(NM_FILE_NAME, 0)

' Set destination work directory
TaskRecord.ValueAsString(NM_DIRECTORY) =
SecondRec.ValueAsString(NM_DIRECTORY, 0)

If Not (WorkObject Is Nothing) Then

' Get operation object depending on specific SmClass and operation name

OperName = NM_OPER_CHECKOUT

Set Operation = Metainfo.OperationsForClass(WorkObject.ClassId,
False).ItemByName(OperName)

' Check if operation allowed for object
```

```
    If SessionUtil.OperationAllowedOnObject(WorkObject, Operation, False)
Then
    ' Perform operation on object

    Set NewWorkObject = SessionUtil.HandleCheckOutNewRelease(WorkObject,
Operation, TaskRecord)

    ' Pass new checked out object to ST through ThirdRec.

    ThirdRec.CopySmRecord NewWorkObject.Data, 0

    CONV_ComRecListToRecordList    ThirdRec, ThirdPar

End If

End If

HandleCheckOut = Err_None

End Function
```

Individual Lifecycle Task: Creating a new Part Revision

The following script uses the NewPartRevision function to create a new Part Revision from a source object.

```
Dim SourceObject As ISmObject

Dim NewPartRevision As ISmObject

Dim PartAttributes As ISmRecord

Set PartAttributes = GetDefaultNewRevisionPartAttributes(SourceObject )

Comment = "New Part Revision"

Set NewPartRevision = NewPartRevision(SourceObject, PartAttributes , Comment
)
```

Individual Lifecycle Task: Promoting a Part object

The following script uses the ExecuteOperationOnObjectTree function to promote a Part object.

```
'Display old state
```

```
MsgBox WorkObject.State

OperName = 'PROMOTE'

Set Operation = Metainfo.OperationsForClass(WorkObject.ClassId,
False).ItemByName(OperName)

Set DefaultTask=Nothing

'Add effectivity dates to tasks - this is important for Promote operation -
part objects are not File managed

Propagate = False

' Set main operation name

OperName = 'PROMOTE'

Set MainOperation = Metainfo.OperationsForClass(WorkObject.ClassId,
False).ItemByName(OperName)

Set TaskRL=CreateObject("SmRecList.SmRecordList")

TaskRL.AddHeader NM_CLASS_ID, 2, sdtSmallInt

TaskRL.AddHeader NM_OBJECT_ID,SIZE_OBJ_ID,sdtInteger

TaskRL.AddHeader NM_DSC_NOTES, 256, sdtChar

j = 0

TaskRL.ValueAsSmallInt(NM_CLASS_ID,j) = WorkObject.ClassId

TaskRL.ValueAsInteger(NM_OBJECT_ID,j) = WorkObject.ObjectId

TaskRL.ValueAsString(NM_DSC_NOTES,j) = "Promote TaskRL notes"

If Not (WorkObject Is Nothing) Then

    ' Check if operation allowed for object

    If SessionUtil.OperationAllowedOnObject(WorkObject, Operation,False) Then

        ' Perform operation on object using SmSessionUtil method

        SessionUtil.ExecuteOperationOnObjectTree WorkObject, MainOperation,
Propagate, TaskRL, DefaultTask

    End If

End If

'Display new state
```

```
WorkObject.Retrieve  
MsgBox WorkObject.State
```

Individual Life-Cycle Operation Task: Replacing Previous Revision While Keeping Check Out

The 'Before Check in' handles the file name, is hooked to the following script when changing the task- NM_LFCYC_CHECKIN_MODE

```
If Sstr <> NM_OPER_CHECKIN Then Exit Function  
  
Dim SmObj As ISmObject, PrevObj As ISmObject  
Set SmObj = Session.ObjectStore.ObjectFromData(RecList1.GetRecord(0), True)  
  
Dim Util As SmSessionUtil  
Set Util = Session.GetService("SmUtil.SmSessionUtil")  
  
Set PrevObj = Util.GetObjectByRevision(SmObj,  
SmObj.Data.Value(NM_PAR_REVISION))  
  
If SmObj.ObjectId = PrevObj.ObjectId Then Exit Function ' if no previous  
revision exists  
  
Dim filename As String, name As String, ext As String, pos As Integer  
filename = SmObj.Data.Value(NM_FILE_NAME)  
pos = InStr(filename, ".")  
name = Left(filename, pos - 1)  
ext = Right(filename, Len(filename) - pos)  
  
If RecList2.Value(NM_LFCYC_CHECKIN_MODE, 0) = LFCYC_WorkRev Then  
  
RecList2.Value(NM_LFCYC_CHECKIN_MODE, 0) = LFCYC_PrevRev ' In 'Replace  
previous revision' we have to provide the FILE_NAME task of previous object  
  
RecList2.Value(NM_FILE_NAME, 0) = PrevObj.Data.Value(NM_FILE_NAME)  
Else
```

```
RecList2.Value(NM_LFCYC_CHECKIN_MODE, 0) = LFCYC_WorkRev ' ' In 'Current
revision' we have to build the FILE_NAME task of current object

RecList2.Value(NM_FILE_NAME, 0) = name & "_" & SmObj.ObjectId & "_" &
SmObj.ClassId & "." & ext

End If
```

Group Life-Cycle Operations

- These functions perform life-cycle operations on groups of objects.
The user can define advanced preferences for the executing operation.

Function	Description	Calls scripts: <u>O</u> ptionally <u>A</u> lways <u>N</u> ever
ExecuteOperationOnObjectTree	Perform life-cycle operation object tree	A
ExecuteOperationOnTrees	Perform life-cycle operation set of object trees	A
GroupUndoCheckOut	Undo group check out. Alwa calls script.	A
GroupUndoCheckOutEx	Undo group check out. Can behavior preference to not c scripts.	O

Understanding Group Life-Cycle Operations

In order to use the group life-cycle methods effectively, you need to understand the way that **SmarTeam** carries out a group lifecycle operation.

There are two types of group life-cycle operations:

A life-cycle operation performed on a single object tree with a single root
– corresponding to the method `ExecuteOperationOnObjectTree`.

A life-cycle operation performed on more than one object tree, each with a
single root -- corresponding to the method `ExecuteOperationOnTrees`.

Lifecycle Operation on a Single Object Tree

This section describes how to perform a life-cycle operation on a single object tree using the function:

```
Sub ExecuteOperationOnObjectTree (SmObject As ISmObject,  
    'Leading object SmOperation As ISmOperation , 'Life-cycle operation  
    Propagated As Boolean, 'To propagate operation to children  
    ObjectAndTreeTasks As ISmRecordList, 'Task record list  
    DefaultTasks As ISmRecordList 'Default tasks record list)
```

A later section describes how to perform life-cycle operations on multiple trees. The basic concepts are the same for both; they are described in detail in this section.

To understand this operation, you need to know:

What are the elements of the object tree on which the life-cycle operation is performed?

Which life-cycle operations are performed?

How to specify task attributes for each life-cycle operation that is performed

Object Tree – Root Object and Descendents

An object tree is composed of a selected root object or leading object together with the linked objects and CFOs, of the root object. For example, an object tree could be an assembly together with all of its parts, subparts and raw materials, or a folder together with all of its documents.

The root object is selected by the user, for example, on a **SmarTeam** View, and the hierarchical descendents are added to the tree automatically by **SmarTeam**. **SmarTeam** retrieves the descendents from the database using the links to the root object. Note that a descendent will not be retrieved if the user is not authorized to retrieve it.

Figure 15 Object Tree Record List

	Object Attributes				
Header: Name Type Size	Oper_ID sdtSmallInt 2	Object_ID sdtObject Identifier 4	Class_ID sdtSmallInt 2	...	Attribute-n Type-n Size-n
RootObject-1	OperID-1	ObjectID-1	ClassID	...	Value-n-1
ChildObject-2	OperID-2	ObjectID-2	ClassID	...	Value-n-2
ChildObject-3	OperID-3	ObjectID-3	ClassID	...	Value-n-3
...
ChildObject-m	OperID-m	ObjectID-m	ClassID	...	Value-n-m

Lifecycle Operations Performed on Objects

One attribute in the object record list is the life-cycle operation OperId for the object. For the Lifecycle Stage 2 script hook, the value in this column is the Operation Code for the operation rather than the Operation Id used elsewhere. A specific main life-cycle operation is specified for the root object and that operation is always used for the root object. However, the lifecycle operation that is performed on the descendent elements of the object tree can be different than the main operation.

Table 4 Operation Code Values

Operation Name	Operation Code	Description
OPCHECKOUT	0	CHECKOUT
OPNEWREL	1	NEWREL
OPCHECKIN	3	CHECKIN
OPAPPROVE	4	APPROVE
OPFREEZE	5	FREEZE
OPCOPYFILE	6	COPYFILE
OPNOOP	7	NOOP
OPNOTALLOWED ¹	8	NOTALLOWED
OPDUMMY	9	DUMMY
OPSECONDARYCOPY	10	SECONDARYCOPY
OPLOCKCOPY	11	LOCKCOPY
OPUNLOCK	12	UNLOCK

Normally, the same operation is used on all descendent elements even if it is not the main operation. However, by using a script, you can cause different operations to be performed on different descendents by changing the value of the Operation Code for that object.

The operation that is actually performed on the descendent elements of an object tree depends on the following factors:

The possible operations that can be performed on the descendants is limited and determined by the selection of main operation.

The main operation can be propagated to the descendents by setting the NM_PROPAGATED and NM_PROPAGATED_IDENT attributes in the object record list. For the ExecuteOperationOnObjectTree method these flags are set by the Propagated parameter. For the ExecuteOperationOnTrees method you set the flags directly in the object record list.

If the main operation is not propagated (because the flag is not set), a **SmarTeam** default operation is used for the descendents. The default operation for each main operation is shown in the table below.

The user can intervene in a script hook to change the default operation on a descendant to one of the other permitted operations.

¹ The operation codes from this row until the end of the table are read-only.

The following table shows the operations permitted on the descendants for each main operation as well as the default operation used on the descendants in case the main operation is not propagated.

Main Operation	Operations Permitted on Descendants	Default for no propagation
CheckIn	CheckIn, Registr, Approve	No Operation
Approve	Approve, Registr, CheckIn	No Operation
CheckOut	CheckOut, NewRel, CopyFile	CopyFile
NewRel	NewRel, CheckOut, CopyFile	CopyFile
CopyFile	CopyFile, SecondaryCopy, LockCopy	CopyFile
LockCopy	LockCopy, CopyFile, SecondaryCopy	
DisableFlowSecurity	DisableFlowSecurity, DisableFlowSharing	
DisableFlowSharing	DisableFlowSharing, DisableFlowSecurity	
EnableFlowSecurity	EnableFlowSecurity, EnableFlowSharing	
EnableFlowSharing	EnableFlowSharing, EnableFlowSecurity	
Freeze	Freeze, NoOp	NoOp
NoOp	NoOp, NotAllowed, Dummy, ReturnNewObject	
NotAllowed ²	NotAllowed, NoOp, Dummy, ReturnNewObject	
Dummy	Dummy, NoOp, NotAllowed, ReturnNewObject	
UndoCheckOut	UndoCheckOut	UndoCheckOut

Specifying Task Attributes for an Operation

In the previous section, we saw that more than one life-cycle operation can be used on the elements of a tree: the main operation on the root element and possibly other permitted operations on each of the descendants.

This section describes how to specify task attributes for each of these operations. There are three ways you can do this:

You can specify task attributes for an operation performed on a specific object, where you need to specify the Object Id, the Class Id and the task attributes.

² The operations from this row to the end of the table are not available to the user.

You can specify task attributes for a specific operation, where you just need to specify the OperationId and the task attributes. These attributes are used any time the operation acts on an object – except for objects specified by method 1 above.

If you do not specify task attributes in either of the previous ways, the **SmarTeam** default task attributes are used for the operation.

1 - Specifying Task Attributes for an Operation Performed on a Specific Object

This section describes case 1 above. If you know the Object Id, Class Id and the operation to be performed for a specific object, you can use the following record lists to specify task attributes for the operation:

Method	Record List
ExecuteOperationOnObjectTree	ObjectAndTreeTasks
ExecuteOperationOnTrees	Tasks

1.1 - Specifying Task Attributes for the Main Operation Performed on the Root Object

In the simplest case, you can use this record list to specify task attributes for the main operation performed on the root object.

If you want to specify life-cycle operation attributes for the root object only, it is sufficient to supply a record list containing a single record of attributes. You do not need to specify either object or operation information. This format is identical to the Task Record parameter for the individual life-cycle attributes described above. The remaining members of the tree are handled with the default attributes.

Figure 16 Task Record for the Root Object

Life-Cycle Operation Attributes				
Header: Name	Attribute-1	Attribute-2	...	Attribute-n
Type	Type-1	Type-2		Type-n
Size	Size-1	Size-2		Size-n
LC Operation	Value-1	Value-2	...	Value-n

1.2 - Specifying Task Attributes for Additional Objects

If you know the Object Id, Class Id and the operation to be used for specific descendents in the tree, you can use the record list to specify task attributes for the operation. You build a record containing the object information and task attributes appropriate to the operation to be performed on the object.

Figure 17 Task Record List

	Object Attributes		Operation Task Attributes		
Header: Name Type Size	Object_ID 10 4	Class_ID 2 2	Attribute-1 Type-1 Size-1	...	Attribute-n Type-n Size-n
RootObj	ObjectID-0	ClassID-0	Value-1-0	...	Value-n-0
Child-1	ObjectID-1	ClassID-1	Value-1-1	...	Value-n-1
Child-2	ObjectID-2	ClassID-2	Value-1-2	...	Value-n-2
Child-3	ObjectID-3	ClassID-3	Value-1-3	...	Value-n-3
...
Child-m	ObjectID-m	ClassID-m	Value-1-m	...	Value-n-m

For example, if the main operation is CheckOut and you know that Copy File will be used on a specific descendant, you can specify attributes for the Copy File operation for that descendant, such as a destination file name.

You do not need to specify the Operation Id in the task record list (however, if you do not specify the Operation Id in the object record list, you must specify it in the task record list.) The Operation Id is already located in the object record list together with the object as shown above. **SmarTeam** assumes that the task attributes you specify for that object in the task record list are appropriate attributes for the operation on that object.

2 - Specifying Task Attributes for a Specific Operation

This section describes case 2 above. **SmarTeam** prepares a set of default task attributes for each operation permitted on the descendents (see table above). For example, when the main operation is CheckOut, default task attributes are prepared for CheckOut, NewRelease, and CopyFile. When one of these operations is performed on a descendant, and that descendant's object information does not appear in the Task record list, the default task attributes are used for the operation.

You can replace or add to the default task attributes for one of the permitted operations using the following record list.

Method	Record List
ExecuteOperationOnObjectTree	DefaultTasks
ExecuteOperationOnTrees	DefaultTasks

You create a record containing the Operation Id for a permitted operation together with the task attributes you want to add or replace as follows.

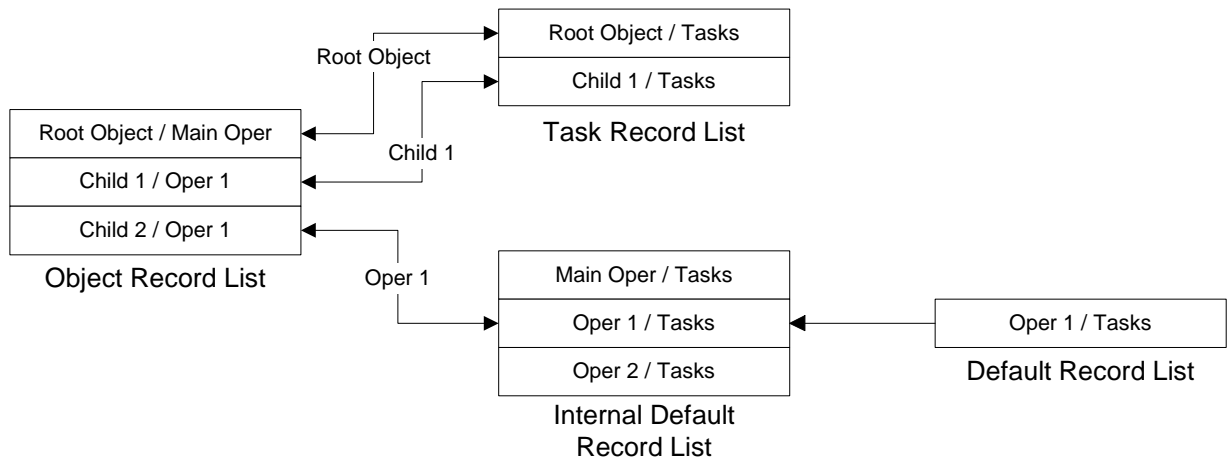
Figure 18 Default Tasks Record List

	Operation	Operation Task Attributes		
Header: Name	Oper_ID	Attribute-1	...	Attribute-n
Type	10	Type-1		Type-n
Size	4	Size-1		Size-n
MainOperation	OperID-0	Value-1-0	...	Value-n-0
PermittedOperation-1	OperID-1	Value-1-1	...	Value-n-1
PermittedOperation-2	OperID-2	Value-1-2	...	Value-n-2
PermittedOperation-3	OperID-3	Value-1-3	...	Value-n-3

You only need to define a record for each operation that you want to change from its default tasks. **SmarTeam** maintains an internal Default Task record list for all permitted operations. The information you provide alters that internal record list for the operations you specify.

Operating with Record Lists

The following figure shows how **SmarTeam** uses the record lists Task and Default in executing life-cycle operations on a tree. **SmarTeam** cycles through the Object Record List. For each object in the Object Record List, it searches for task information for the object's operation. First, it looks for the object's identifying information in the Task Record List (like Root object and Child 1 in the figure). If found, SmarTeam uses the task information from the Task Record List. If not, it finds the object's operation in the Default Record List (Child 2 in the figure) and uses the task information from there.



Note: You should not alter any of the values of Operation Code in the Life-Cycle Stage 2 hook.

Example

This example checks out the parent assembly, uses Task Record List to copy a specific child part files to *E:\Program Files\View*, and uses Default Record List to copy the other child parts to *D:\smartsolutions\examples*.

```
` create record for Copy File operation in Default task

OperName = NM_OPER_COPY_FILE

Set Operation = Metainfo.OperationsForClass(WorkObject.ClassId,
False).ItemByName(OperName)
```



```
Set DefaultTask=CreateObject("SmRecList.SmRecordList")

DefaultTask.AddHeader NM_OPER_ID, 2, sdtSmallInt

DefaultTask.AddHeader NM_DIRECTORY, 256, sdtChar

DefaultTask.ValueAsSmallInt(NM_OPER_ID,0) = Operation.Id

DefaultTask.ValueAsString(NM_DIRECTORY,0) = "D:\smartsolutions\examples"

Propagate = False

' Set main operation name

OperName = NM_OPER_CHECKOUT

Set MainOperation = MetaInfo.OperationsForClass(WorkObject.ClassId,
False).ItemByName(OperName)

Set TaskRL=CreateObject("SmRecList.SmRecordList")

TaskRL.AddHeader NM_CLASS_ID, 2, sdtSmallInt

TaskRL.AddHeader NM_OBJECT_ID,SIZE_OBJ_ID,sdtInteger

TaskRL.AddHeader NM_DSC_NOTES, 256, sdtChar

TaskRL.AddHeader NM_REVISION, 256, sdtChar

TaskRL.AddHeader NM_FILE_NAME, 129, sdtChar

TaskRL.AddHeader NM_DIRECTORY, 256, sdtChar

j = 0

TaskRL.ValueAsSmallInt(NM_CLASS_ID,j) = WorkObject.ClassId

TaskRL.ValueAsInteger(NM_OBJECT_ID,j) = WorkObject.ObjectId

TaskRL.ValueAsString(NM_DSC_NOTES,j) = "NEWRELEASE TaskRL notes"

TaskRL.ValueAsString(NM_REVISION,j) = "x2"

j = 1

' Check if children exist for this object

If Count <> 0 Then

    For i = 0 To Count-1

        Set Child = Children.Item(i)

        If Child.ClassId = PartClassId Then
```

```
        If Child.Data.ValueAsString("CN_ID") = "SWP-0075" Then
            TaskRL.ValueAsSmallInt(NM_CLASS_ID,j) = Child.ClassId
            TaskRL.ValueAsInteger(NM_OBJECT_ID,j) = Child.ObjectId
            TaskRL.ValueAsString(NM_DSC_NOTES,j) = "Copy File TaskRL
Child Document notes"
            TaskRL.ValueAsString(NM_FILE_NAME,j) = "assembled.doc"
            TaskRL.ValueAsString(NM_DIRECTORY,j) = "E:\Program
Files\View"
            TaskRL.ValueAsString(NM_REVISION,j) = "d1"
            j=j+1
        End If
    End If
Next
End If
If Not (WorkObject Is Nothing) Then
    ' Check if operation allowed for object
    If SessionUtil.OperationAllowedOnObject(WorkObject, Operation,False) Then
        ' Perform operation on object using SmSessionUtil method
        SessionUtil.ExecuteOperationOnObjectTree WorkObject, MainOperation,
Propagate, TaskRL, DefaultTask
    End If
End If
```

Multiple-Tree Group Life-Cycle Operations

Life-cycle operations can be performed on a group of trees using the method:

```
Sub ExecuteOperationOnTrees(
    LeadingObjects As ISmObjects,           'Leading object of each tree
    Tasks As ISmRecordList,                 'Task record list
    DefaultTasks As ISmRecordList           'Default task record list
```

In general this method works the same as `ExecuteOperationOnObjectTree`. The advantage of using this function over using `ExecuteOperationOnObjectTree` is its convenience.

The two record lists `Tasks` and `DefaultTasks` work the same as the record lists `ObjectAndTreeTasks` and `DefaultTasks` of the method `ExecuteOperationOnObjectTree`. See the explanation in the previous section.

The parameter `LeadingObjects` contains the root object or leading object of each tree on which you want to perform a life-cycle operation. It replaces the `SmObject` parameter of the method `ExecuteOperationOnObjectTree`.

In addition you need to specify the attributes `NM_OPER_ID` and `NM_PROPAGATED` for each object in `LeadingObjects`. These replace the parameters `SmOperation` and `Propagated` of the method `ExecuteOperationOnObjectTree`.

There are two ways you can specify the attributes `NM_OPER_ID` and `NM_PROPAGATED` for each object:

If you want to specify the same attribute value for all objects, load them into the `DefaultTasks` record list. From there they will be loaded into all the objects in the `LeadingObjects` parameter.

If you want to specify the same or different values for these parameters, load them directly into the `LeadingObjects` parameter and do not load them into the `DefaultTasks` record list.

SmarTeam converts `LeadingObjects` into an `ObjectList` record list and adds all descendants of all root object to the record list.

See example below.

Group Life-Cycle Operation Task: Check In an Object Tree

The following script uses the `ExecuteOperationOnObjectTree` function to check in an object tree. It is defined as a user-defined command.

```
Function CheckInObjectTree(ApplHndl As Long,Sstr As String,FirstPar As  
Long,SecondPar As Long,ThirdPar As Long ) As Integer
```

```
Dim SmSession As SmApplic.SmSession
```

```
Dim FirstRec As Object
```

```
Dim SecondRec As Object

Dim ThirdRec As Object

Dim Operation As SmApplic.ISmOperation 'performed operation object

Dim Metainfo As SmApplic.SmMetaInfo 'metainfo object for smClasses

Dim SessionUtil As SmUtil.SmSessionUtil 'main SmarTeam service object

Dim DefaultTask As Object 'default task record for operation

Dim TaskRL As Object 'task record list for operation per object

Dim OperName As String 'operation name

Dim Propagate As Boolean 'Propagate operation

Dim WorkObject As SmApplic.ISmObject

Dim QueryDefinition As SmApplic.ISmQueryDefinition

Dim Children As SmApplic.ISmObjects ' collection of objects linked

Dim Child As Object '

Dim FolderClassId As Integer

Dim i As Integer

Dim j As Integer

Dim Count As Integer

' use this script for setting defaults in the LC screen through the
ThirdPar and for setting individual records

' through SecondPar

MsgBox Sstr + " CheckInObjectTree"

' ExecuteOperationOnObjectTreeExample =
LFCycBrowseOper(ApplHndl,Sstr,FirstPar,SecondPar,ThirdPar)

' MsgBox "After"

' Exit Function

' Convert pointer to COM object SmSession

Set SmSession = SCREXT_ObjectForInterface(ApplHndl)

Set SessionUtil = SmSession.GetService("SmUtil.SmSessionUtil")

Set Metainfo = SmSession.Metainfo
```

```
FolderClassId = Metainfo.SmClassByName("Folder").ClassId

' Conver input parameter to COM object
CONV_RecListToComRecordList FirstPar,FirstRec

CONV_RecListToComRecordList SecondPar,SecondRec

CONV_RecListToComRecordList ThirdPar,ThirdRec


Set WorkObject =
SmSession.ObjectStore.ObjectFromData(SecondRec.GetRecord(0),true)

' Propagate operation for all of the object's children

' Define query
Set QueryDefinition = Nothing

' Retrieve all object's children
Set Children = WorkObject.RetrieveChildren(QueryDefinition)

Count = Children.Count

Propagate = True

' Set operation name according to default constant
OperName = NM_OPER_CHECKIN

' Get operation object depending on specific SmClass and operation name


Set Operation = Metainfo.OperationsForClass(WorkObject.ClassId,
False).ItemByName(OperName)

'Set TaskRL = Nothing

Set TaskRL=CreateObject("SmRecList.SmRecordList")

TaskRL.AddHeader NM_CLASS_ID, 2, sdtSmallInt 'dont need operid - applies to
leading object

TaskRL.AddHeader NM_OBJECT_ID,SIZE_OBJ_ID,sdtInteger 'dont need operid -
applies to leading object

TaskRL.AddHeader NM_OPER_ID,2, sdtSmallInt 'dont need operid - applies to
leading object

TaskRL.AddHeader NM_LFCYC_CHECKIN_MODE, 2, sdtSmallInt 'dont need operid -
applies to leading object
```

```
TaskRL.AddHeader NM_DSC_NOTES, 256, sdtChar

j = 0

TaskRL.ValueAsSmallInt(NM_CLASS_ID,j) = WorkObject.ClassId

TaskRL.ValueAsInteger(NM_OBJECT_ID,j) = WorkObject.ObjectId

TaskRL.ValueAsSmallInt(NM_OPER_ID,j) = Operation.Id

TaskRL.ValueAsSmallInt(NM_LFCYC_CHECKIN_MODE,j) = 2

TaskRL.ValueAsString(NM_DSC_NOTES,j) = "Checkin TaskRL notes"

j = 1

' Check if children exist for this object

If Count <> 0 Then

    For i = 0 To Count-1

        Set Child = Children.Item(i)

        If Child.ClassId = FolderClassId Then

            TaskRL.ValueAsSmallInt(NM_CLASS_ID,j) = Child.ClassId
            TaskRL.ValueAsInteger(NM_OBJECT_ID,j) = Child.ObjectId
            TaskRL.ValueAsSmallInt(NM_OPER_ID,j) = Operation.Id
            TaskRL.ValueAsSmallInt(NM_LFCYC_CHECKIN_MODE,j) = 2          'check in
            folder as previous

            TaskRL.ValueAsString(NM_DSC_NOTES,j) = "Checkin TaskRL notes"

            j=j+1

        End If

    Next

End If

' Empty task record - default tasks record for operation

Set DefaultTask=CreateObject("SmRecList.SmRecordList") 'create obj from co-
class of library

DefaultTask.AddHeader NM_OPER_ID, 2, sdtSmallInt          'need operid in
DefaultTask - applies to all objects

DefaultTask.ValueAsSmallInt(NM_OPER_ID,0) = Operation.Id

DefaultTask.AddHeader NM_DSC_NOTES, 256, sdtChar
```

```

DefaultTask.ValueAsString(NM_DSC_NOTES,0) = "Checkin DefaultTask notes"

'Set DefaultTask = Nothing
If Not (WorkObject Is Nothing) Then
' Check if operation allowed for object
    If SessionUtil.OperationAllowedOnObject(WorkObject, Operation,False) Then

        ' Perform operation on object using SmSessionUtil method
        SessionUtil.ExecuteOperationOnObjectTree WorkObject, Operation,
        Propagate, TaskRL, DefaultTask

    End If
End If

CONV_ComRecListToRecordList    SecondRec, SecondPar

CheckInObjectTree = Err_None

End Function

```

Group Life-Cycle Operation Task: Check Out a Set of Object Trees

```

Function CheckOutTrees(ApplHndl As Long,Sstr As String,FirstPar As
Long,SecondPar As Long,ThirdPar As Long ) As Integer

Dim SmSession As SmApplic.SmSession

Dim FirstRec As Object

Dim SecondRec As Object

Dim ThirdRec As Object

Dim Operation As SmApplic.ISmOperation 'performed operation object

Dim MetaInfo As SmApplic.SmMetaInfo 'metainfo object for smClasses

Dim SessionUtil As SmUtil.SmSessionUtil 'main SmarTeam service object

Dim DefaultTask As Object 'default task record list for operation

Dim TaskRL As Object 'task record list for operation per object

Dim LeadingObjects As SmApplic.ISmObjects

```

```
Dim OperName As String 'operation name

Dim Propagate As Boolean 'Propagate operation

Dim WorkObject As SmApplic.ISmObject

Dim QueryDefinition As SmApplic.ISmQueryDefinition

Dim Children As SmApplic.ISmObjects ' collection of objects linked

'Dim Child As SmApplic.ISmObject '

Dim Child As Object '

Dim FolderClassId As Integer

Dim LOIndex As Integer

Dim TaskRLIndex As Integer

Dim RecordCount As Integer

Dim ChildrenCount As Integer

Dim k As Integer

Dim Result As Long

' Convert pointer to COM object SmSession

Set SmSession = SCREXT_ObjectForInterface(ApplHndl)

Set SessionUtil = SmSession.GetService("SmUtil.SmSessionUtil")

Set Metainfo = SmSession.Metainfo

FolderClassId = Metainfo.SmClassByName("Folder").ClassId

' Create SmObjects and Rec Lists

Set LeadingObjects = SmSession.ObjectStore.NewObjects

LeadingObjects.Data.AddHeader NM_OPER_ID, 2, sdtSmallInt

LeadingObjects.Data.AddHeader NM_PROPAGATED, 2, sdtSmallInt

'Task Record List

Set TaskRL=CreateObject("SmRecList.SmRecordList")

TaskRL.AddHeader NM_CLASS_ID, 2, sdtSmallInt

TaskRL.AddHeader NM_OBJECT_ID, SIZE_OBJ_ID, sdtInteger

TaskRL.AddHeader NM_OPER_ID,2, sdtSmallInt
```



```
TaskRL.AddHeader NM_DSC_NOTES, 256, sdtChar

TaskRL.AddHeader NM_REVISION, 256, sdtChar

'Default Record List

'create obj from co-class of library
Set DefaultTask=CreateObject("SmRecList.SmRecordList")

DefaultTask.AddHeader NM_PROPAGATED, 2, sdtSmallInt

DefaultTask.AddHeader NM_DSC_NOTES, 256, sdtChar

' Conver input parameter to COM object
CONV_RecListToComRecordList FirstPar,FirstRec

CONV_RecListToComRecordList SecondPar,SecondRec

CONV_RecListToComRecordList ThirdPar,ThirdRec


OperName = NM_OPER_CHECKOUT

Set Operation = MetaInfo.SmOperationByName(OperName)

' Default tasks record for operation - necessary
DefaultTask.ValueAsSmallInt(NM_PROPAGATED,0) = 1

DefaultTask.ValueAsString(NM_DSC_NOTES,0) = "Checkout DefaultTask notes"


RecordCount = FirstRec.RecordCount

If RecordCount <> 0 Then

    TaskRLIndex = 0

    For LOIndex = 0 To RecordCount-1 'loop over leading objects

        Set WorkObject =
SmSession.ObjectStore.ObjectFromData(FirstRec.GetRecord(LOIndex),true)

        Result = LeadingObjects.Add(WorkObject)

        Set Operation =
MetaInfo.OperationsForClass(LeadingObjects.Item(LOIndex).ClassId,
False).ItemByName(OperName)

        LeadingObjects.Data.ValueAsSmallInt(NM_OPER_ID,LOIndex) =
Operation.Id
```

```
'fill task list for leading object

TaskRL.ValueAsSmallInt(NM_CLASS_ID,TaskRLIndex) = WorkObject.ClassId

TaskRL.ValueAsInteger(NM_OBJECT_ID,TaskRLIndex) = WorkObject.ObjectId

TaskRL.ValueAsString(NM_DSC_NOTES,TaskRLIndex) = "Checkout TaskRL
notes"

TaskRL.ValueAsString(NM_REVISION,TaskRLIndex) = "y"

TaskRLIndex = TaskRLIndex + 1

' Retrieve all object's children

Set QueryDefinition = Nothing

Set Children = WorkObject.RetrieveChildren(QueryDefinition)

ChildrenCount = Children.Count

' fill task list for children - same operation as parent

If ChildrenCount <> 0 Then

    For k = 0 To ChildrenCount-1

        Set Child = Children.Item(k)

        'check in folders with previous revision

        If Child.ClassId = FolderClassId Then

            TaskRL.ValueAsSmallInt(NM_CLASS_ID,TaskRLIndex) =
Child.ClassId

            TaskRL.ValueAsInteger(NM_OBJECT_ID,TaskRLIndex) =
Child.ObjectId

            TaskRL.ValueAsString(NM_DSC_NOTES,TaskRLIndex) =
"Checkout TaskRL notes"

            TaskRL.ValueAsString(NM_REVISION,TaskRLIndex) = "w"

            TaskRLIndex=TaskRLIndex + 1

        End If

    Next

End If
```

```
        Next
    End If
    'Set TaskRL = Nothing
    If Not (LeadingObjects Is Nothing) Then
        SessionUtil.ExecuteOperationOnTrees LeadingObjects, TaskRL, DefaultTask
    End If
    CheckOutTrees = Err_None
End Function
```

Lifecycle Authorization Operations

The SmSessionUtil object contains the following functions to check authorization for life-cycle operations.

Function	Description
OperationAllowedForStateAndClass	Determines if a life-cycle operation is allowed a life-cycle state.
OperationAllowedOnObject	Determines if a life-cycle operation is allowed the current life-cycle state of an object.
OperationAllowedOnObjectAndAuthorized	Determines if a life-cycle operation is allowed the current life-cycle state of an object and if is authorized for the operation and the class.
GetTargetState	Get resulting life-cycle state of an object after undergoing the life-cycle operation.

Common Tasks

See examples of the life-cycle operations above for examples of the use of these functions.

Mask Operations

This section describes the mask operations in the SmUtil object.

Note: The functionality of the mask operations has been incorporated in the new SmSequence object. It is recommended to use it instead of the functions listed in this section.

The SmSessionUtil object contains the following methods to handle masks:

Function	Description
MaskRollBack	Rollbacks a specified attribute mask to a given value.
MaskTruncate	Returns a truncated attribute mask according to modified group and value.
RetrieveMaskGroupCount	Returns number of groups in a specified attribute mask.
RetrieveNextMask	Given a specific class attribute, increments and retrieves the mask value/
RetrieveNextRevision	Increments the revision mask and returns it.
RetrieveStartMaskValue	Given specific class attribute, retrieves the mask initial value.

Miscellaneous Utilities

The SmSessionUtil object contains the following special persistent object retrieval functionality:

Function	Description
CurrentSessionPreference	Given section name and preference name, retrieves the value, for example, the current format of time stamp value.
GetObjectByRevision	Given a specific object's revision number, retrieves the appropriate persistent object.
GetObjectsByCadIdentity	Given FileName, Directory and class, retrieves all corresponding persistent objects.
TranslateToReturnCode	Translates a standard COM error code to one of the SmarTeam -specific error types.
GetObjectStateIcon	Gets the icon for the object state.
GetOptionValue	Get current session preference.

Common Tasks

```
Sub Test

    Dim TimeStampFormat as string

    TimeStampFormat = SmSessionUtil. CurrentSessionPreference("CONVERSION
FORMATS", " TIMESTAMP")

    MsgBox "Current TimeStamp format is ..." + TimeStampFormat

End Sub
```

SmMiscUtil Object

The **SmMiscUtil** object provides various utility functions. Unlike the **SmSessionUtil** object, the functions provided by this object are not dependent on the working session, database or user.

The object contains the following functionality:

- **SmConvert** – retrieves an **SmConvert** object (See **SmConvert** object)

SmConvert and SmSessionConvert Objects

The **SmConvert** object provides various conversion functions. The **SmSessionConvert** object provides a subset of the functions provided by **SmConvert**. Unlike **SmConvert**, **SmSessionConvert** performs the conversion according to the **SmarTeam** preference formats.

SmConvert Object

The object contains functionality for converting string variables into specific type variables and the opposite.

The conversion functionality divides into two groups:

- Simple conversion functionality , such as:
- `IntegerValueAsWideString`
- `IntegerValueFromWideString`
- `DoubleValueAsWideString`
- `DoubleValueFromWideString`
- Advanced conversion functionality, in which the user passes a string format that should be designed as follows:

To convert string variables into specific type variable, the string variable should be structured according to the input format. For example:

```
DateVal = DateValueFromWideString("22/02/98",Format)
```

The Format should be "dd/mm/yy" in order for DateVal to contain the appropriate value.

To convert specific type variables into a string variable, the output would be structured according to the input format. For example :

```
StringVal = DateValueAsWideString(DateVal,"dd\mm\yy")
```

The variable dateVal is a date variable. If its value is September 1st 1999, the StringVal would be "01\09\99"

- **IsValueEmpty** – checks whether specific variable contains an empty value
- **SetEmptyValue** – retrieves a specific type empty value.

SmSessionConvert Object

The object contains a subset of the advanced conversion functionality of SmConvert, but using the session formats and not an input format (as in SmConvert)

Example

```
DateVal = DateValueFromWideString(StringVal)
```

StringVal must be structured according to the session Date preference value.

```
StringVal = DateValueAsWideString(DateVal)
```

8. SmarTeam - Workflow Library

General Description

The **SmarTeam - Workflow** library enables you to perform the following workflow-related functions:

- Flow process functionality:
- Initialize process
- Attach objects to the Flow process
- Attach a flowchart to the Flow process
- Send a Flow process
- Maintain of a list of queues, including the standard Incoming, Sent, and Deleted queues.
- FlowSession functionality:
- Get Information about the Current User
- Get Information about the User Queues
- Work with the FlowQueue Collection
- FlowQueueItem functionality:
- Get Information about the Workplace Environment
- Get Information about the FlowQueueItem
- Select Objects Linked to the FlowQueueItem
- Get the Status of the FlowQueueItem
- Capture a FlowQueueItem
- Check Task Results
- Delegate Users to the FlowQueueItem
- Add Run-Time Users to a Node.
- ActiveProcess functionality:
- Get Information about the ActiveProcess
- Execute Tasks
- Send the FlowProcess
- Send an E-Mail Ahead
- Work with the FlowSentProcesses Collection

Overview of Objects

You can access the Flow Services through the SmFlowStore object, which is obtained by the mechanism of services accessed through the session.

A SmFlowStore object is obtained as follows:

```
Dim FlowStore As SmartFlow.FlowStore  
  
Set FlowStore = SmSession.GetService("SmartFlow.FlowStore")
```

This section presents an overview of the main **SmarTeam - Workflow** objects including a description of the associated objects that are useful for the programmer:

- SmFlowProcess Object
- SmFlowChart Object
- SmFlowSession Object
- SmWorkflowView Object
- SmFlowStore Object

SmFlowProcess Object

Description

An **SmFlowProcess** object represents a planned sequence of work activities on a selected set of **SmarTeam** objects. The SmFlowProcess includes an initial state where work on the objects begins, a final state where work on the objects is finished, and intermediate “workplaces” where users can perform tasks on the objects of the FlowProcess.

The possible sequences that the FlowProcess can take between workplaces must conform to a predetermined plan – represented by paths on a Flowchart (described below). The actual routing of a particular FlowProcess between workplaces on the Flowchart is determined by the decisions of the users at each workplace at the time the FlowProcess executes.

FlowProcess Classes

Several different classes of FlowProcess objects can be defined in the **SmarTeam - Workflow** system. The classes correspond to types of work processes normally encountered in industry, such as ECP, ECO, and Engineering Release processes.

Flowchart Object

The **SmFlowchart** object represents the plan of the SmFlowProcess. It includes all possible workplaces (nodes) and routing paths (connectors) a FlowProcess can visit. As mentioned, an actual FlowProcess will take one specific path through the Flowchart, depending on the responses of the users. See page 120 for more information about the SmFlowchart object.

Selecting a Flowchart for a FlowProcess

Each FlowProcess class can have a set of Flowcharts assigned to it. One Flowchart can be designated as the default Flowchart for that class. When a FlowProcess object is created from a FlowProcess class for a particular application, an appropriate Flowchart is selected from the set of Flowcharts assigned to the class and is attached to the FlowProcess object.

The **SmProcessAssignment** object supports the assignments of Flowcharts to FlowProcess classes. One **SmProcessAssignment** object is defined for each FlowProcess class and represents the collection of Flowchart objects that are assigned to that FlowProcess class. The **SmProcessAssignment** object provides methods to add and remove Flowcharts object assignments and access the default Flowchart. See page 104 for more information about the FlowProcess object.

The **SmProcessAssignments** object is the collection of SmProcessAssignment objects for all FlowProcess classes.

Process History

The **SmProcessHistory** object is a record of the work activity on a FlowProcess. Each time the FlowProcess passes a Node a separate ProcessHistoryItem entry is recorded. The information includes the Executors and the Responses at the Node.

Note: If you log out of a service and plan to log back into a service, before logging out, you must close the service as follows:

```
Session.Services.Close("SmartFlow.SmFlowStore");
```

Object Diagram

The object diagram of SmFlowProcess is shown below:

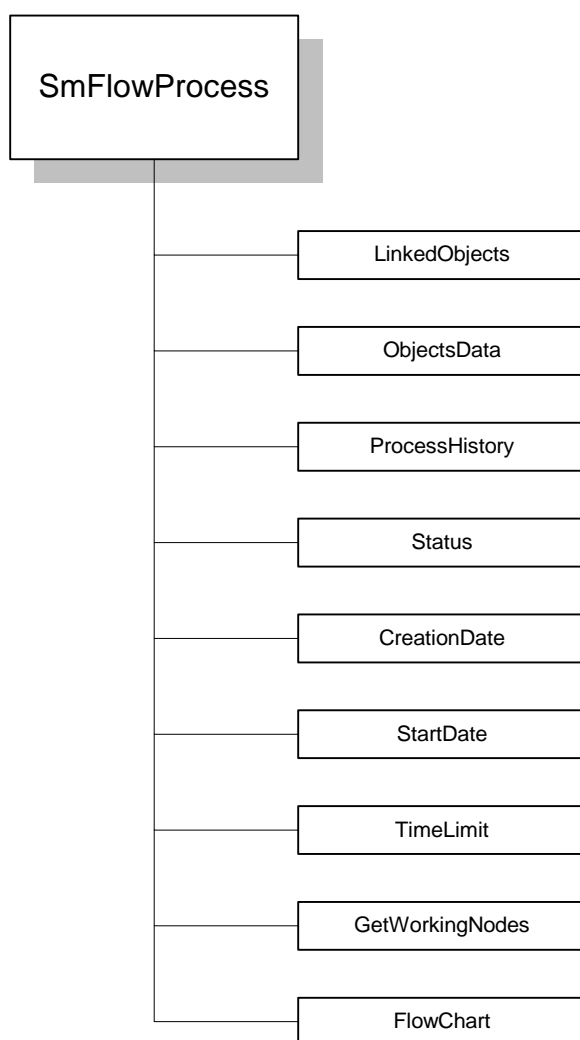


Figure 8-1 FlowProcess Object Diagram

Obtaining an Attached Object

Use the following to attach an object.

```
Function BeforeAttachObject(FlowProcess As Object,  
SmObject As Object,  
ProcessContents As Object) As Integer
```

Obtaining the SmFlowProcess Object

1. For a stand-alone application:

```
Set SmEngine = CreateObject("SmApplic.SmEngine")  
  
SmEngine.Init "SmTeam32"  
  
Set SmSession = SmEngine.CreateSession("Test Session", "Smart32")  
  
Set FlowStore = SmSession.GetService("SmartFlow.SmFlowStore")  
  
ProcessClassId = SmSession.MetaInfo.SmClassByName("General Process").ClassId  
  
Set FlowProcess = FlowStore.InitiateNewProcess(ProcessClassId)
```

2. In an event or task-driven script use the FlowProcess or ActiveProcess parameters.

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to a FlowProcess.

FlowProcess Task: Getting the FlowProcess Attributes

Field Name	Description
TDM_DESCRIPTION	Description of FlowProcess
TDM_END_TIME	Time of the process termination
TDM_STATUS	Reference to "TDM_SF_PROCESS_STATUS"
TDM_IMPORTANCE	Reference to "TDM_IMPORTANCE"
TDM_TIME_LIMIT	Time limit of process
TDM_SECONDARY_LNK	0 – do not create secondary links automatically, 1 – create automatically

FlowProcess Task: Getting the FlowProcess Execution Status

A FlowProcess can be in one of four execution states. The following table describes the states and the software constant used for each state.

State	Description	Software Constant
Initiated	The FlowProcess has been initiated and has not been sent from the Start Node	fpsInitiated
Running	The FlowProcess has been sent from the Start Node and has not reached the End Node	fpsRun
Ended	The FlowProcess has reached the End Node	fpsEnded
Terminated	The FlowProcess has been terminated by a user is in the CompletedItems queue.	fpsTerminated

Use the **Status** property to get the execution status of a FlowProcess.

Example

```
If FlowProcess.Status = fpsInitiated Then
    MsgBox ("Process is in Initiated state")
End If
```

FlowProcess Task: Getting the Time Limits of the FlowProcess

The **CreationDate** property gives the time when the process was created

The **StartDate** property gives the time when the initiator sent the process from the Start Node. If the initiator did not send the process yet, its value is null.

The **EndDate** property gives the time the process reached the End Node. If the process has not yet reached the End Node, its value is null.

The **DueDate** gives the time the process is due to be completed. If the process has not yet been sent by the initiator, its value is null.

Example

```
If Date > FlowProcess.DueDate Then
```

```
MsgBox ("Process is overdue")  
  
End If
```

The **TimeLimit** is the elapsed time during which the process should be completed. In case the FlowProcess is finished in time, the EndDate will be equal or less than StartDate plus TimeLimit.

FlowProcess Task: Getting the Execution Stage of the FlowProcess

Use the **GetWorkingNodes** to get a collection of all Nodes whose Queue contains the current FlowProcess. This tells you at which stage of workflow the process is located.

Example

```
Dim FlowProcess As SmartFlow.SmFlowProcess  
  
MsgBox "Number of working nodes for process: " &  
CStr(FlowProcess.GetWorkingNodes.Count)
```

FlowProcess Task: Getting Objects Linked to the FlowProcess

Use the **LinkedObjects** property to get a MultiCompositeObject object that represents the objects that are linked to the current FlowProcess. The links themselves are in the object. Each member of the collection is a pair of elements: the first element is the link between object and FlowProcess and the second element is the object itself.

The **ObjectsData** property gets the collection of objects without the links. **ObjectData** has the type IsmMultiObjects. IsmMultiObjects is a collection of collections: that is, a collection whose members are collections. The member collections are collections of objects from various super classes that are attached to the current FlowProcess, such as the Documents, Items and Users collections.

Example

```
' An Active Process was obtained either as a parameter in a SmarTeam script  
or after capturing a FlowProcess.  
  
' get all objects linked to Active Process without their links
```



```
Set LinkedObjects = ActiveProcess.FlowProcess.ObjectsData
' LinkedObjects has type IsmMultiObjects:
' including super classes Documents, Items, Users

For I = 0 To LinkedObjects.Count - 1
    ' create object for one superclass
    Set SubLinkedObjects = LinkedObjects(I)
    ' loop on that superclass collection ;lkj lkj lkj l;kj ;lkj lkj ;lkj
    For J = 0 To SubLinkedObjects.Count - 1
        . . .
```

FlowProcess Task: Getting Flowchart Properties

You get the properties of the FlowChart (or the FlowChart template for an initiated FlowProcess) attached to the current FlowProcess using the **FlowChart** property.

FlowProcess Task: Creating a FlowProcess Object

When you create a new FlowProcess object, you need to attach an appropriate Flowchart. Each type of FlowProcess has Flowchart types that are suited to it including a default Flowchart.

To attach the default Flowchart to the newly created FlowProcess object, use **AttachDefaultFlowchart**. To attach a specified Flowchart, use **AttachFlowchart**.

To send the FlowProcess from the Start Node, use the **InitiateProcess** method.

Example

```
Set Node = FlowProcess.Flowchart.StartNode
' create a response object from the response of the first out connector
Set Response = Node.OutConnectors.Item(0).Response
' send FlowProcess from start node on all connectors with that Response
Comment = "Auto sent process"
FlowProcess.InitiateProcess FlowSession, Response, Comment, Date
```

Changing Flowchart Properties at the Start Node

When you create a new FlowProcess object and attach a Flowchart to it, initially only a skeleton Flowchart is attached and a pointer is established to the Flowchart template. The full Flowchart is copied from the template only when the FlowProcess is sent from the Start Node.

Therefore, if you want to make changes to a new FlowProcess object at the Start Node that affect the attached Flowchart, you need to use the **FullFlowChartCopy** method first to copy the entire Flowchart from its template. An example of a change that affects the Flowchart is adding new users to subsequent nodes of the Flowchart.

If you do not make such changes you do not need to perform the **FullFlowChartCopy** method.

FlowProcess Task: Deleting a FlowProcess Object

A User can delete a FlowProcess only while it is in the Initiated status, that is, at time of initialization at the Start Node. Otherwise, only a Supervisor or Database Administrator can delete it.

FlowProcess Task: Linking Objects to a FlowProcess

You can use the **LinkObject** method to attach an object, for example, a Document, to a FlowProcess. The method creates a link object belonging to Complex Link class "TDM_SF_PROCESS_CONTENTS".

Example

```
Set FlowProcess = FlowStore.InitiateNewProcess(ProcessClassId)
' establish a complex link between ObjectToSend and the FlowProcess
Set LinkAttributes = Nothing
Set AttachedObject = FlowProcess.LinkObject(ObjectToSend, LinkAttributes)
```

Primary and Secondary Links between FlowProcess and Object

Links between an object and a FlowProcess can be Primary links or Secondary links. Primary links are the links that the user creates between an object and the FlowProcess, for example, using the **LinkObject** method. In the event that the object has children, dependencies or CFO objects (for example, an assembly), **SmarTeam - Workflow** can create additional—Secondary—links between them and the FlowProcess.

These Secondary links are created automatically when the user creates a Primary link to the object if the following attributes have been set to TRUE in the Flowchart object:

Secondary Links to:	Property
Children of Primary-linked object	ShouldLinkChildren
Dependencies of Primary-linked object	ShouldLinkDependencies
CFOs (Common File Object) of Primary-linked obj	ShouldLinkMOOFs

You can specify directly that an object be linked to a process by a secondary link using the **LinkObjectAsSecondary** method. An object linked with a secondary link does not appear on the process view at the first level.

Example

```
' A Document represents an ISmObject
Set LinkAttributes = Nothing

' secondary link document to process; doesn't appear on process view
Set Link = FlowProcess.LinkObjectAsSecondary(Document,LinkAttributes)
```

Use the **UnlinkObject** method to detach an object from a FlowProcess

FlowProcess Task: Controlling Sharing of Objects between Processes

The **ShareObjects** property of a FlowProcess controls whether an object attached to the FlowProcess can be attached to a second FlowProcess. If the **ShareObjects** property of a FlowProcess is set to `osLimiting`, once an object has been attached to that FlowProcess, it cannot be linked to a second FlowProcess, even if the **ShareObjects** property of the second FlowProcess is set to `osNotLimiting`.

FlowProcess Task: Controlling the Security Level

The **SecurityLevel** property of a FlowProcess controls which users can perform operations on objects attached to the FlowProcess.

If the **SecurityLevel** property is set to `slvHighSecurity`, only users that are permitted to act through the FlowProcess can execute operations on objects attached to the FlowProcess. For example, if a Document were attached to a FlowProcess, **SmarTeam** users not permitted to work through the FlowProcess would not be able to work on that Document at all, even through other **SmarTeam** windows.

If the **SecurityLevel** property is set to `slvAttachedToFlowProcess`, any **SmarTeam** user can perform operations on objects attached to the current FlowProcess, subject to the authorization mechanism of **SmarTeam – Editor**.

Example

```
If FlowProcess.SecurityLevel = slvHighSecurity Then  
    MsgBox "High security flow process"  
End If
```

ProcessHistory Task: Working with the ProcessHistory Collection

The ProcessHistory collection contains a set of ProcessHistoryItem objects. It represents the entire history of a FlowProcess, broken down into states of the FlowProcess at each Send-from-Node event.

Use the **Item** property to get a ProcessHistoryItem from the collection

Example

` Retrieves the first record from the Process History.

```
Set ProcessHistoryItem = FlowProcess.ProcessHistory.Item(0)
```

Use the **Refresh** method to refresh the ProcessHistory collection.

ProcessHistoryItem Task: Get the ProcessHistoryItem Information

SmProcessHistoryItem is an element of the ProcessHistory collection.

The ProcessHistoryItem contains information about the FlowProcess state at a single Send-from-Node event. Thus the ProcessHistoryItem refers to a specific Node, Executor, and Response.

Note that if the FlowProcess is sent again from the same Node, even with the same Executor and Response, it is considered to be a different Send-from-Node event and so a new ProcessHistoryItem is generated.

A single ProcessHistoryItem includes the elements:

- The Node from which the FlowProcess was sent
- The ProcessHistoryInformation at the Node for a Send-from-Node event

Use the **Node** property to get the Node.

Use the **History** property to get the ProcessHistoryInformation from the ProcessHistoryItem.

```
Set ProcessHistoryInformation = ProcessHistoryItem.History
```

ProcessHistoryInformation Task: Get Historical Information about this FlowProcess

This section contains methods and properties to extract historical information about the state of the FlowProcess at a specific Send-from-Node event.

Use the **ReceiveTime** property to get the time the FlowProcess was received at this Node.

Use the **ActualProcessingTime** property to get the elapsed time from when the User captured the FlowProcess until it was sent from this Node.

Use the **StartTime** property to get the time when the user captured the FlowProcess.

Use the **EndTime** property to get time the FlowProcess was sent from this node.

The **Importance** property of the ProcessHistoryInformation object is defined as the maximum value of the Importance property of the FlowProcess itself and the Importance property of the FlowProcess at this Node.

Use the **Response** property to get the Response on which the FlowProcess was sent from the Node.

Use the **Comment** property to get the comment that was entered when the FlowProcess was sent from this Node.

Use the **PastDueFlowchart** property to determine if the FlowProcess was sent from this Node after the FlowProcess time limit was exceeded.

Use the **Executor** property to get the User that worked on the FlowProcess for this Node.

Note that if another User worked on the same FlowProcess at the same Node, that information would be stored in a separate ProcessHistoryItem.

Example

```
Dim History As SmartFlow.SmProcessHistoryInformation
Dim FlowStore As SmartFlow.SmFlowStore
Dim FlowQueueItem As SmartFlow.SmFlowQueueItem

Set FlowStore = SmSession.GetService("SmartFlow.SmFlowStore")
Set FlowQueueItem = FlowStore.FlowSession.InboxProcesses(0)
'Get first history information record
Set History = FlowQueueItem.FlowProcess.ProcessHistory.Item(0).History
' Get the UserLogin of the user that started this process
UserLogin = History.Executor.Data.ValueAsString("LOGIN")
StartDate = CStr(History.StartTime)
```

MsgBox "The process initialized by user " & UserLogin & " at: " & StartDate

SmFlowChart Object

Description

The **SmFlowchart** object represents a work plan for a FlowProcess. A Flowchart is composed of nodes and connectors. A node can be considered as a “workplace” where users can perform tasks on the objects linked to the FlowProcess. The connectors between the nodes define the possible sequencing or routing of the FlowProcess between the workplaces – the order in which work can be performed on the FlowProcess.

In general, a Flowchart can be designed with several connectors exiting a node in order to take into account all possible ways the user at the node may decide to dispose of the FlowProcess. Each connector corresponds to one possible response of the user at the node. When the FlowProcess is executed, its actual path through the Flowchart is determined by the responses of the users at each node.

The Flowchart specifies:

- The users that can work on the FlowProcess at each node
- The tasks the users can perform at each node
- The connector paths that the FlowProcess can take through the nodes, depending on the response of the user.

SmFlowchart Object

SmFlowchart properties provide references to the Flowchart components as follows:

- FlowProcess –the FlowProcess to which the Flowchart is attached
- Nodes – the set of Nodes on the Flowchart
- Connectors – the set of Connectors on the Flowchart
- Supervisor – the Supervisor, a user with special privileges on the Flowchart

The **SmFlowCharts** object represents a collection of **SmFlowChart** objects.

SmNode Object

The **SmNode** object represents an individual node of the Flowchart. The **SmNode** properties provide access to the Node components as follows:

- **Tasks** – the tasks to be performed on the Node. An individual task at a node is represented by a **SmTask** object. The **SmTask** object TaskType property specifies whether the task is manual, operation-based, or runs a script.
- **Executors** – the **SmarTeam** users defined as users for the node. An individual user at a Node is represented by the **SmExecutor** composite object consisting of a link and a **SmarTeam** User object. The link associates the Node with the User object, designating that **SmarTeam** User to be an Executor of the Node. The User object contains the data about the user.
- **InConnectors, OutConnectors** – the incoming and outgoing connectors from this Node

SmConnector Object

The **SmConnector** object represents an individual connector. **SmConnector** properties provide references to the Connector components as follows:

- **Response** – the response of the Connector. A response is represented by the **SmResponse** object. The **SmResponse** object has a ResponseType property indicating whether the type of the response is Reject or Accept.
- **FromNode, ToNode** – the source and destination Nodes of the Connector.

The **SmConnectors** object represents a collection of **SmConnector** objects.

Object Diagram

The object diagram of SmFlowChart is shown below:

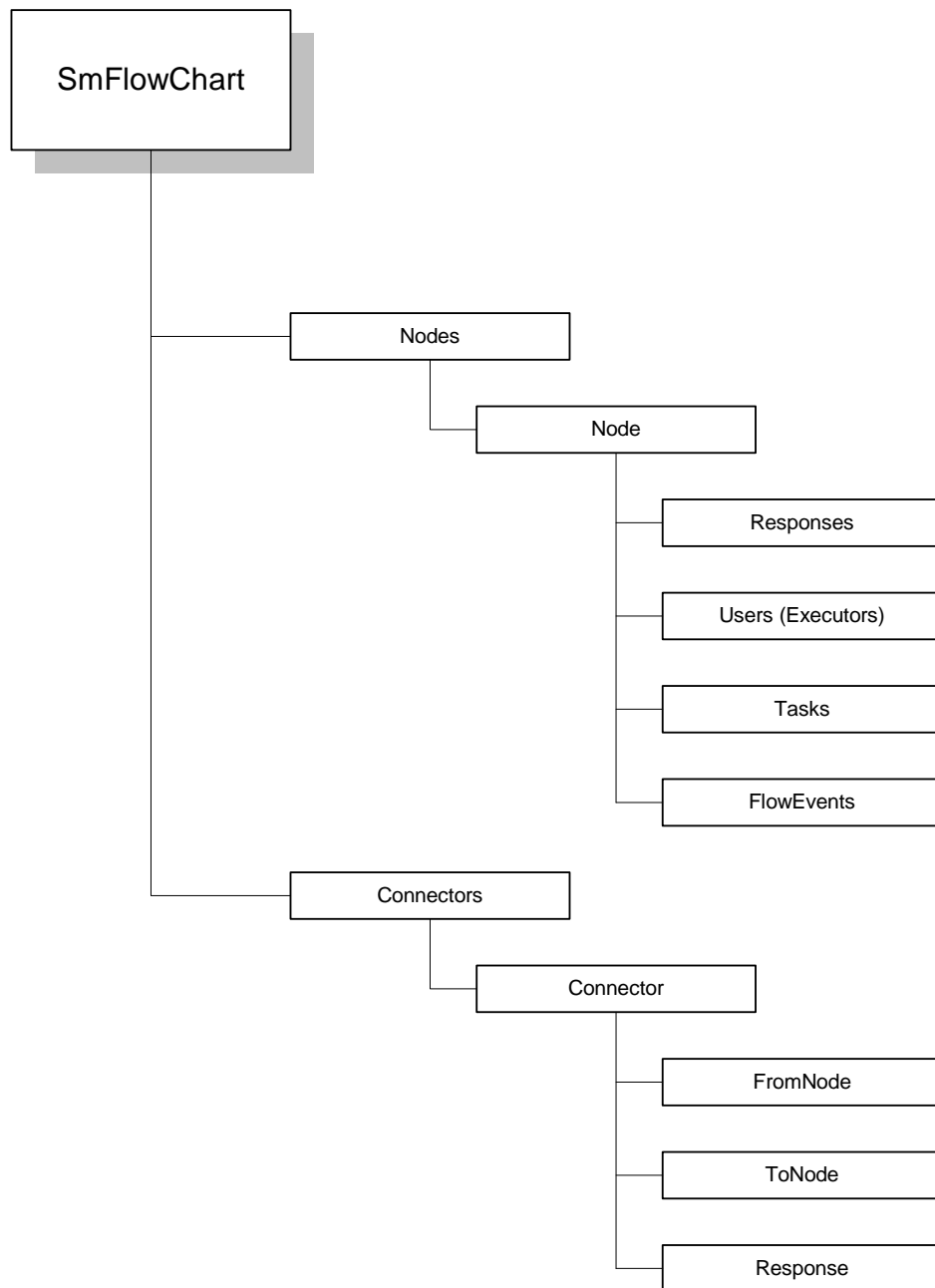


Figure 8-2 *FlowChart* Object Diagram

Obtaining the SmFlowChart Objects

A Flowchart object is obtained from a FlowProcess object:

```
ProcessClassId = SmSession.MetaInfo.SmClassByName("General Process").ClassId
```

```
Set FlowProcess = FlowStore.InitiateNewProcess(ProcessClassId)
```

```
Set Flowchart = FlowProcess.Flowchart
```

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to a Flowchart and its components.

FlowChart Task:

Getting Information about the Flowchart.

Use the **Nodes** property to get a collection of all Nodes in flowchart.

```
Set Nodes = Flowchart.Nodes
```

Use the **Connectors** property to get a collection of all connectors in flowchart.

Use the **StartNode** property to reference the Start node

Example.

```
' obtain node object representing Start Node
```

```
Set Node = FlowProcess.Flowchart.StartNode
```

Use the **EndNode** property to reference the End node.

Use the **FlowchartType** property to get the Flowchart type. The Flowchart types are described in the following table.

Flowchart Type	Description	Software Constant
Template	The Flowchart is an original template as created in Flow Designer	ftTemplate
Library	Collection of pre-designed Nodes stored in Library (not relevant for writing script)	ftLibrary
Work Copy	Copy of original template for attaching to FlowProcess	ftWorkCopy
Dummy	The Flowchart is a dummy Flowchart, which is used for a FlowProcess in its Initiated state.	ftDummy

Use the **Supervisor** property to get the User object that is defined as the Supervisor. You can use **Supervisor** property to designate a Supervisor in the Initiated state of the FlowProcess.

For more information about the properties: **ShareObjects**, **ShouldLinkChildren**, **ShouldLinkDependencies**, **ShouldLinkMOOFs**, **SecurityLevel** see the corresponding properties in the FlowProcess interface.

Nodes Task: Get Individual Nodes from a Collection

The following example shows how to get an individual Node from a Nodes collection.

Example

```
Set Node = Nodes.Item(0) ' get the first node
```

Node Task: Getting Information about the Node.

This section describes methods and properties you use when you work with a Node object. In real time, the Node object occurs in the context of a Flowchart attached to a specific FlowProcess. Thus, the Node can have Users and Tasks associated with it through its FlowProcess.

Use the **Tasks** property to get an ISmTasks collection that represents the tasks defined at this Node.

Use the **Users** property to get an ISmExecutors collection that represents the possible Executors at this Node.

```
Set Users = FromNodes(i).Users.GetUsers
```

Use the **InConnectors** property to get SmConnectors collection that represents the entering connectors.

Use the **OutConnectors** property to get an SmConnectors collection that represents the exiting connectors.

Example

' create a response object from the response of the first out connector

```
Set Response = Node.OutConnectors.Item(0).Response
```

Use the **GetPreviousNodesSentToCurrent** method to get an SmNodes collection of the previous Nodes that sent a FlowProcess associated with this node's Flowchart to the current Node.

Use the **GetFollowingNodesEmailList** method to get a semicolon-delimited SmStrings list of e-mail addresses of all users in the following Nodes.

Use the **NodeType** property to get the Node type. The Node types are described in the following table:

Node Type	Description	Software Constant
Start	The Node is the Start Node	ntStart
End	The Node is the End Node	ntEnd
User Defined	The Node is a user-defined node, defined in Flow Designer	ntUserDefined
Automated	not implemented	ntAutomated
Information	Displays Flowchart title or other text. Not for receiving FlowProcess	ntInformation

Use the **GetNotRejectResponses** method to get an SmResponses collection that represents all responses, defined on connectors that exit this Node, that are not of type Reject.

Example

```
ProcessClassId = SmSession.MetaInfo.SmClassByName  
("General Process").ClassId  
  
' create and initiate new process, attach default Flowchart  
  
Set Process = FlowStore.InitiateNewProcess(ProcessClassId)  
  
Set Node = Process.Flowchart.StartNode  
  
' select the first non-reject response for start node  
  
Set ProcessResponse = Node.GetNotRejectResponses(0)
```

Use the **GetRejectResponses** method to get an SmResponses collection that represents all responses, defined on connectors that exit this Node, that are of type Reject.

Use the **FlowStatus** property to get the flow status at the Node. The flow status is the status of the Node relative to the FlowProcess. The following table describes the flow status:

FlowStatus	Description	Software Constant
Pending	The FlowProcess arrived at the Node but the Users have taken no action.	nsPending
Completed	The FlowProcess has been sent from this Node	nsCompleted
Locked	This Node is locked. No User can perform actions. can occur if a User rejected the FlowProcess to any Node preceding this Node.	nsLocked
Captured	The FlowProcess has been captured at this Node.	NsCaptured
Inactive	The FlowProcess has not reached this Node, or can never reach it.	nsInactive

Use the **Policy** property to get the flow policy at this Node. The flow policy specifies the way in which users are allowed to capture a FlowProcess at this Node, as described in the following table.

Policy	Description	Software Constant
And	More than one User can capture the FlowProcess at this Node and all Users are required to perform the tasks defined at this Node.	fpAnd
Or	Only one User can capture a FlowProcess at this Node and that User is required to perform the tasks defined at this Node.	fpOr

Use the **Delegate** property to check whether there is a Delegator defined in this node.

Note: You can use this property as a read property only

Use the **Value** property to get the value of a Node attribute. The following table shows some of the properties you can get only through the **Value** property.

Field Name	Description
TDM_IS_TEMPLATE	Whether task/flow definition is template or actual instance
TDM_IMPORTANCE	Reference to "TDM_IMPORTANCE"
TDM_TIME_LIMIT	Time limitation of task or linked process
TDM_PROCESS_ID	Reference to "TDM_SF_PROCESS". Used only in the copies the process. In the templates it must be NULL_OBJ_ID
TDM_OBLIG_NODE	Node is obligatory
TDM_LOOP_COUNT	Count of times node has been reached

Node Task: Saving the Node.

The **Save** method saves the Node, together with its Users and Tasks, to the Database.

Node Task: Changing Users

Use the **ReplaceExecutor** method to replace an existing Executor with another one at this Node.

Use the **RuntimeUsers** property to check if an Executor at an immediately previous Node or at the Start Node can specify an Executor at this Node.

Example

```
Dim FlowStore As SmartFlow.SmFlowStore

Dim FlowQueueItem As SmartFlow.SmFlowQueueItem

Dim DestinationNode As SmartFlow.SmNode

Dim Connector As SmartFlow.SmConnector


Set FlowStore = SmSession.GetService("SmartFlow.SmFlowStore")

Set FlowQueueItem = FlowStore.FlowSession.InboxProcesses(0)

Set Connector = FlowQueueItem.Node.OutConnectors.ItemByName("Forward to
Designer")

Set DestinationNode = Connector.ToNode


' check if new users can be added to the Destination Node
If DestinationNode.RuntimeUsers Then

' loop over selected users
    For I = 0 To Users.Count - 1
        ' add selected users to destination node
        FlowQueueItem.AddRunTimeUsers DestinationNode, Users(0)
    Next
End If
```

Use the **CanChangeUsers** method to check if the User of a specified ActiveProcess can select Executors at this Node at runtime. The method checks if the Node of the ActiveProcess immediately precedes the current Node or if it is the Start Node.

Use the **CanTryChangeUsers** method to check if the User of a specified FlowQueueItem can select Executors at this Node at runtime. The method checks if the Node of the FlowQueueItem immediately precedes the current Node or it is the Start Node. Note that Executors can be specified only if the FlowQueueItem is successfully captured.

SmTask Task: Getting Information about the Task.

This section describes methods and properties you use when you work with a Task object. In real time, the Task object occurs in the context of a specific Node of a Flowchart attached to a specific FlowProcess.

Example

```
Dim ActiveProcess As SmartFlow.SmActiveProcess

'Show script name for specific process task

MsgBox ActiveProcess.CurrentNode.Tasks.Item(0).ScriptName
```

Use the **Value** property to get the value of a Task attribute. The following table contains some of the Task attributes that can be obtained only through the **Value** property.

Field Name	Description
TDM_PROCESS_ID	Reference to FlowProcess. Used only in the actual copies of the process; in the definitions it must be NULL_OBJ_ID
TDM_PER_OBJECT	Whether task should be performed on the process objects or on process itself
TDM_AUTOMATED	Reference to internal Lookup TDM_SF_TASK_AUTOMATED

Use the **TaskType** property to get the task type. The task type specifies how the task is to be performed, as described in the following table:

FlowStatus	Description	Software Constant
Manual	The task is performed manually.	ttManual
Operation	The task activates a pre-selected lifecycle operation.	ttOperation
Script	The task activates a script.	ttScript

If the task is an Operation task, use the **OperationID** property to get its operation ID.

If the task is a Script task, use the **ScriptName** property to get the script name of the script that is activated by the task.

Use the **Required** property to check if the task must be performed in order that the associated FlowProcess continue.

SmTask Task: Executing a Task

Use the **Execute** method to execute the current Task. The objects on which the Task operates depends on the setting of PerObject, as follows:

- If PerObject was selected, then the Task is executed for all objects that were selected by the FlowQueueItem SelectedObjects property.
- If PerObject was not selected, the Task is executed on all objects attached to the FlowProcess. Results are returned as IsmOperationResults and are stored in the database.

See the ActiveProcess Task: Executing Tasks section for an alternate way of executing tasks.

Example

'This example uses methods and properties:

' CanCapture, TaskResultStatus, Execute, PendingStatus and others

```
Sub Main()  
  
    Dim Smart As SmarTeam.SmApplication  
  
    Dim SmEngine As SmApplic.SmEngine  
  
    Dim Session As SmApplic.SmSession  
  
  
    Set Smart = GetObject(, "SmarTeam.SmApplication")  
  
    Set SmEngine = Smart.Engine  
  
    Set Session = SmEngine.Sessions(0)  
  
    Test Session  
  
End Sub
```

```
Sub Test(SmSession As SmApplic.SmSession)  
  
    Dim FlowStore As SmartFlow.SmFlowStore  
  
    Dim FlowSession As SmartFlow.SmFlowSession  
  
    Dim FlowQueueItem As SmartFlow.SmFlowQueueItem
```

```

Dim ActiveProcess As SmartFlow.SmActiveProcess

Dim Task As SmartFlow.SmTask

Dim OperationResults As SmApplic.SmOperationResults

' Get FlowStore object - Sm Flow Service
Set FlowStore = SmSession.GetService("SmartFlow.SmFlowStore")

' Get current user flow session
Set FlowSession = FlowStore.FlowSession

' Check if exist process in Inbox of smartbox
If FlowSession.InboxProcesses.Count > 0 Then
    ' Retrieve first process from user inbox
    Set FlowQueueItem = FlowSession.InboxProcesses.Item(0)

    ' Check if process has not performed tasks
    If FlowQueueItem.AppropriateTasks.Count > 0 Then
        ' Check if process can be captured
        If FlowQueueItem.CanCapture Or FlowQueueItem.PendingStatus =
fisCapture Then
            ' Capture process
            FlowQueueItem.Capture

            ' Get active process
            Set ActiveProcess = FlowQueueItem.ActiveProcess

            For I = 0 To FlowQueueItem.AppropriateTasks.Count - 1
                ' Get task to execute
                MsgBox I

                Set Task = FlowQueueItem.AppropriateTasks.Item(I)

                ' Check if task not performed yet
                If FlowQueueItem.TaskResultStatus(Task) = rsOperNotPerformed
Then
                    ' Perform task and get results per objects

```

```
        Set OperationResults = Task.Execute(ActiveProcess)

    End If

Next

End If

End If

End If

End Sub
```

Use the **GetObjectResultStatus** method to get the result status for a task performed by a specified Executor on a specified object. The Executor input parameter is provided for the case that the AND policy for the task is in effect and more than one Executor can perform the task on the same specified object.

The results can be:

Result Status	Description	Software Constant
Task not performed	The task was not performed.	rsOperNotPerformed
Task performed successfully	The task succeeded.	rsOperExecutedSuccessfully
Task failed	The task failed.	rsOperExecutedNotSuccessfully
Not appropriate	unused	rsOperNotAppropriate

Connector Task: Getting Information about the Connector

Use the **ToNode** property to get the destination Node of the Connector.

Use the **FromNode** property to get the source Node of the Connector.

Use the **Response** property to get the Response of the Connector

Example

' create a response object from the response of the first out connector

```
Set Response = Node.OutConnectors.Item(0).Response
```

Use the **Value** property to get an attribute of the Connector. The following table contains some of the Connector attributes that can be obtained only through the **Value** property.

Field Name	Description
TDM_FLOWCHART_ID	Reference to "TDM_SF_FLOWCHART"
TDM_PROCESS_ID	Reference to "TDM_SF_PROCESS". Used only in the actual copie the process; in the definitions it must be NULL_OBJ_ID

Response Task: Getting Information about the Response

Use the **Name** property to get the name of the Response.
Note: Use for read only.

Use the **ResponseType** property to get the Response Type of the current Response.
Note: Use for read only.

Response Type	Description	Software Constant
Accept	The Response has type Accept.	rtAccept
AcceptConsult (not currently in use)	The Response has type Accept Consult	rtAcceptConsult
Reject	The Response has type Reject	rtReject

Example

```
' Find the first response of type "accept"

Set Response = Nothing

Set Responses = FlowStore.Responses

For i = 0 To Responses.Count-1

    If (Responses(i).ResponseType = rtAccept) Then

        Set Response = Responses(i)

        Exit For

    End If

Next i
```

Use the **GetPredefinedId** method to get the value of the pre-defined response property of a Response object.

A pre-defined Response is a Response that is pre-defined in the system and cannot be assigned by the user to a specific Connector. A pre-defined Response can be used by an Executor at any Node. Any Response defined by the user is, by definition, not pre-defined and can be assigned to a connector.

The pre-defined responses are shown in the following table.

Pre-Defined Response	Description	Software Constant
NotPredefined	This Response was defined by user and can be assigned to a connector.	prNotPredefined
RejectToStart	Reject process from current node to start node. This response is predefined in system. It cannot be assigned to a specific connector.	prRejectToStart
RejectToPrevious	Reject process from current node to all immediately previous nodes. This response is predefined in system. It cannot be assigned to a specific connector.	prRejectToPrevious
Consult	(not currently in use)	prConsult
ConsultAndWait	(not currently in use)	prConsultAndWait
Reply	(not currently in use)	prReply

Executors Task: Working with the Executors Collection

The **Executors** collection contains the Executors assigned to this Node (see data model table TDM_SF_EXECUTORS in Appendix).

Use the **Item** property to get an individual Executor by index.

Use the **GetUsers** method to refer to the User objects component of the Executors.

Example

```
Dim FlowStore As SmartFlow.SmFlowStore

Dim FlowQueueItem As SmartFlow.SmFlowQueueItem

Dim FromNodes As SmartFlow.SmNodes


Set FlowStore = SmSession.GetService("SmartFlow.SmFlowStore")

Set FlowQueueItem = FlowStore.FlowSession.InboxProcesses(0)

Set FromNodes = FlowQueueItem.FromNodes

For I = 0 To FromNodes.Count - 1

    ' retrieve collection of users on all From Nodes

    Set Users = FromNodes(I).Users.GetUsers

    ' loop on users of a From Node

    For J = 0 To Users.Count - 1

        ' create duplicate User object to get all attributes for that
        ' user (can't do that when the User is in a collection)

        Set CurrentUser = Users(J).Clone

        ' get info from data base

        CurrentUser.Retrieve

        ' add the User email address from the database data record
        ' to the mail recipient collection

        Mail.Recipients.Add CurrentUser.Data.ValueAsString("USER_EMAIL")

    Next

Next
```

Use the **Delegator** property to get the Executor object corresponding to the Delegator user at this Node, if a Delegator exists.

Executor Task: Getting Information about an Executor

Use the **UserData** property to get the User object component of this Executor

Use the **ExecutorData** property to get the link component of the this Executor

Use the **Delegator** property to check if this Executor is a delegator

Use the **PendingStatus** property to get the work status of this Executor on the process associated with the Flowchart at this Node.

FlowPending Status	Description	Software Constant
New	The Executor did not yet begin work on the Process at this node.	fisNew
Decline	The Executor declined to work on the Process at this Node	fisDecline
Capture	The Executor has captured the Process at this Node	fisCapture
Completed	The Executor has completed working on the Process at this Node and has sent the Process from the Node.	fisCompleted

SmFlowSession Object

Description

The **FlowSession** object represents the work environment of a **SmarTeam** User working with the SmartBox. The FlowSession object allows a **SmarTeam** user to view and access any FlowProcess at any Node at which he is defined as an Executor of the FlowProcess.

The FlowSession object presents the FlowProcesses in the following object categories:

- **InBoxProcesses**
- **SentProcessesCompleted**
- **SentProcessesNotCompleted**
- **SentProcessDeleted**.

The **InBoxProcesses** object is a collection of **FlowProcesses** received at the user's **InBox**. A **FlowProcess** appears in a user's **InBox** when it arrives at a **Node** at which the user is defined as an **Executor** for that **FlowProcess**. The **FlowProcess** remains in the **InBox** for that **Node** until the user (or another user in case the **OR** policy is in effect) sends it from the **Node**.

The **SentProcessesCompleted** object is a collection of processes that have been sent by the user and have been completed, that is, reached the **End Node**.

The **SentProcessesNotCompleted** object is a collection of processes that have been sent by the user and are not yet completed.

The **SentProcessDeleted** object is a collection of processes that have been deleted by the user.

The **FlowQueueItem** object represents an individual process received at the user's **InBox**. The **FlowQueueItem** has the following elements:

- The associated **FlowProcess**
- The current **Node**
- The current **Executor**
- **ProcessLocation** (link between **FlowProcess** and **Node**)

Note: It is important to keep in mind the distinction between the objects **FlowProcess** and **FlowQueueItem**. The former refers to an entire **FlowProcess** and the latter represents the **FlowProcess** as an individual **Executor** receives it at a specific **Node** for action or viewing. Several different **SmFlowQueueItem** objects can represent the same **FlowProcess**, each at the **InBox** of a different **Executor**.

The **ActiveProcess** object represents the **FlowProcess** associated with the **FlowQueueItem** at the current **Node**. You use the **ActiveProcess** to execute tasks defined for the **FlowProcess** and to send the **FlowProcess** from the current node.

Object Diagram

The object diagram of **SmFlowSession** is shown below:

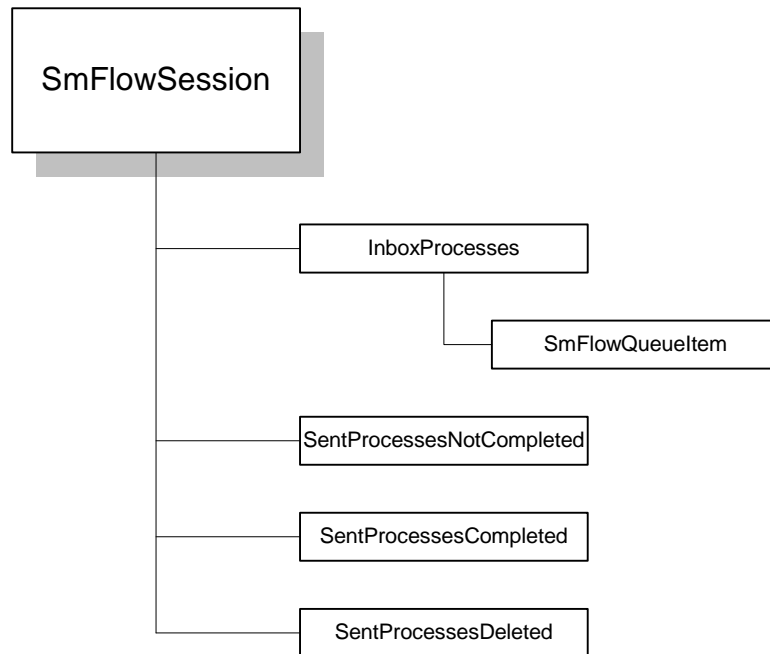


Figure 8-3 FlowSession Object Diagram

Obtaining a SmFlowSession Object

1. For a stand-alone application:

```
Set FlowStore = SmSession.GetService("SmartFlow.SmFlowStore")
```

```
Set FlowSession = FlowStore.FlowSession
```

2. In an event or task-driven script where ActiveProcess is a parameter:

```
Set FlowStore = ActiveProcess.FlowStore
```

```
Set FlowSession = ActiveProcess.FlowSession
```

3. In an event or task-driven script where FlowSession is a parameter use it directly.

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to a FlowSession and its components.

FlowSession Task: Getting Information about the Current User

Use the **User** property to get information about the user associated with the FlowSession.

Example

```
Dim FlowSession As SmartFlow.SmFlowSession

MsgBox "Current user login: " &
FlowSession.User.Data.ValueAsString("LOGIN")
```

FlowSession Task: Getting Information about the User Queues

Use the following FlowSession properties to access the current user queue collections.

Property	Queue Description
InboxProcesses	Collection of Inbox Queue items of user
SentProcessesNotCompleted	Collection of SentProcess items of this user which are yet completed
SentProcessesCompleted	Collection of SentProcess items of this user which are completed
SentProcessesDeleted	A Process can arrive at this queue after being deleted at the queue SentProcessesNotCompleted or SentProcessesCompleted

FlowQueue Task: Working with the FlowQueue Collection

The **FlowQueue** collection represents the set of FlowQueueItem objects associated with a single **SmarTeam** user. This section describes how to work with the FlowQueue collection.

To get an individual FlowQueueItem from the FlowQueue collection by index, use the **Item** property.

To get the index of a specified FlowQueueItem in the collection, use the **IndexOf** method.

Example

```
Dim FlowQueue As SmartFlow.SmFlowQueue  
Index = FlowQueue.IndexOf(FlowQueueItem)
```

Use the **Refresh** method to refresh the FlowQueue to include new FlowQueueItem objects.

To get a FlowQueueItem at a specified Node, use the **ItemByNode** property. There can be at most one FlowQueueItem per Node in the FlowQueue collection.

FlowQueueItem Task: Getting Information about the Workplace Environment

Use the **FlowProcess** property to get the FlowProcess related to this FlowQueueItem. The **Node** property gets the Node object for this FlowQueueItem.

Example

```
Dim FlowStore As SmartFlow.SmFlowStore  
Dim FlowQueueItem As SmartFlow.SmFlowQueueItem  
Dim Node As SmartFlow.SmNode  
  
Set FlowStore = SmSession.GetService("SmartFlow.SmFlowStore")  
Set FlowQueueItem = FlowStore.FlowSession.InboxProcesses(0)  
Set Node = FlowQueueItem.Node  
MsgBox "I am working at node: " & Node.Name
```

FlowQueueItem Task: Getting Information about the FlowQueueItem

The **IsRead** property indicates whether the current FlowQueueItem has been viewed.

The **GetCapturedUsers** method gets a collection of users that have already captured the current FlowQueueItem on this node.

Example

```
Dim FlowStore As SmartFlow.SmFlowStore

Dim FlowQueueItem As SmartFlow.SmFlowQueueItem

Set FlowStore = SmSession.GetService("SmartFlow.SmFlowStore")

Set FlowQueueItem = FlowStore.FlowSession.SentProcessesNotCompleted(0)

Set WorkingUsers = FlowQueueItem.GetCapturedUsers
```

The **Importance** property indicates the level of importance that has been assigned to the current FlowQueueItem.

Importance	Description	Software Constant
High	The FlowQueueItem is very important	fiiHigh
Low	The FlowQueueItem is has low importance	fiiLow
Normal	The FlowQueueItem has normal importance	fiiNormal

The **PastDueNode** property indicates whether the time limit has expired for action on this FlowQueueItem at this node.

The **PastDueFlowchart** property indicates whether the time limit has expired for action on this FlowQueueItem on this Flowchart.

Example

```
Dim FlowQueueItem As SmartFlow.SmFlowQueueItem

If FlowQueueItem.PastDueFlowchart Then

    MsgBox "Flow process past limited time"

End If
```

The **Initiated** property indicates whether the current FlowQueueItem was initiated by current user (executor) and was not yet sent from the Start Node.

Use the **FromNodes** property to get the collection of nodes that sent the current FlowQueueItem.

Example

```
Set FromNodes = ActiveProcess.QueueItem.FromNodes
```

The **ReceiveTime** property gives the time the current FlowQueueItem arrived at the current Node.

The **StartTime** property gives the time the current executor captured the current FlowQueueItem.

The **ExecutorData** property gets information about the current executor.

FlowQueueItem Task: Selecting Objects Linked to the FlowQueueItem

The **SelectedTasks** property gives the collection of tasks, which were selected from collection of all linked objects. This property prepares objects for performing tasks by the ActiveProcess.ExecuteSelectedTasks method.

The **SelectedObjects** property gives the collection of linked objects, which were selected from collection of all linked objects. This property prepares objects for performing tasks by the ActiveProcess.ExecuteSelectedTasks method.

The **SelectAllObjects** method selects all objects linked to the FlowProcess represented by this FlowQueueItem.

The **AppropriateTasks** property gives a list of tasks, which have not yet been executed for the selected objects for this node.

See the “ActiveProcess Task: Executing Tasks” section for an example that uses these methods and properties.

FlowQueueItem Task: Getting the Status of the FlowQueueItem

A FlowQueueItem can be in one of four execution states. The following table describes the states and the software constant used for each state.

State	Description	Software Constant
New	The FlowQueueItem is new	fisNew
Decline	The User has declined to work on the FlowQueueItem	fisDecline
Capture	The FlowQueueItem has been captured	fisCapture
Completed	The FlowQueueItem has been sent	fisCompleted

The **PendingStatus** property gets the status of the FlowQueueItem.

ResetStatus lets you reset the status of the FlowQueueItem to New from Capture or Decline.

FlowQueueItem Task: Capturing a FlowQueueItem

The methods and properties in this section support the capturing of a FlowQueueItem. You need to capture the FlowQueueItem in order to work with its associated FlowProcess, execute its Tasks and send on the FlowProcess.

Use the **CanCapture** property to check whether the AND/OR capture policy allows the current user to capture the FlowQueueItem at this Node. For example, if the policy is OR and another user has already captured the FlowQueueItem, the current user is not allowed to capture it.

To capture the FlowQueueItem only, without working with its associated FlowProcess, use the **Capture** method.

To capture the FlowQueueItem in order to work with its associated FlowProcess, use the **Accept** method. The Accept method returns an ActiveProcess object that you can use to perform tasks associated with the FlowQueueItem and to send the process from this node.

After using the Accept method to capture the FlowQueueItem and create an ActiveProcess object, you can use the **ActiveProcess** property to refer to the ActiveProcess object.

Use the **Decline** method to notify the Supervisor that the current user declines to work on the FlowQueueItem.

FlowQueueItem Task: Checking Task Results

Use the **ObjectResultStatus** property to determine:

1. If a specific task defined for the current FlowQueueItem has already been performed for a specific object
2. The order in which tasks defined for the current FlowQueueItem are performed

Example

```
' check if task (FlowTask) execution for object (AttachedObject) was  
successfully
```



```
' can be used for error treat procedure

If FlowTask.ObjectResultStatus(AttachedObject, UserObject) =
rsOperExecutedNotSuccessfully Then

    MsgBox "Error execution task " + FlowTask.Name + " for specific object"

End If
```

FlowQueueItem Task: Delegating Users to the FlowQueueItem

If a Delegator has been established, the **Delegate** method assigns the FlowQueueItem to the ToUsers list for handling. Users in the ToUsers list can capture the FlowQueueItem according to the AND/OR policy determined by the Delegator through the Policy parameter of the method.

If no Delegator has been established, this method is not applicable.

FlowQueueItem Task: Adding Run-Time Users to a Node

Use the method **AddRunTimeUsers** to add a specified User to a specified target Node.

The method works under the following conditions:

1. The capture policy allows the current user to capture the FlowQueueItem at the current node
2. The current node is immediate predecessor of the target Node – or else the current node is the Start Node
3. The target node is designated as “select user at run time”

ActiveProcess Task: Getting Information about the ActiveProcess

The ActiveProcess object represents the FlowProcess associated with the FlowQueueItem at the current Node. You use the ActiveProcess to execute tasks defined for the FlowProcess and to send the FlowProcess from the current node.

Use the **QueueItem** property to reference the FlowQueueItem associated with the ActiveProcess.

Use the **FlowProcess** property to reference the FlowProcess associated with the ActiveProcess.

Use the **CurrentNode** property to reference the current Node.

```
Set CurrentNode = ActiveProcess.CurrentNode
```

ActiveProcess Task: Executing Tasks

Use the **ExecuteSelectedTasks** method to execute the tasks that were selected by FlowQueueItem.SelectedTasks on the objects that were selected by FlowQueueItem.SelectedObject.

See the “SmTask Task: Executing a Task” section for an alternate way of executing a task.

Example

*'This example uses the methods SelectAllObjects, SelectedTasks,
' ExecuteSelectedTask to executes tasks and send a process.*

```
Sub Main()  
  
    Dim Smart As Object  
  
    Dim SmEngine As SmApplic.SmEngine  
  
    Dim Session As SmApplic.SmSession  
  
  
    Set Smart = GetObject(, "SmarTeam.SmApplication")  
  
    Set SmEngine = Smart.Engine  
  
    Set Session = SmEngine.Sessions(0)  
  
    Test Session  
  
End Sub
```

```
Sub Test(SmSession As SmApplic.SmSession)  
  
    Dim FlowStore As SmartFlow.SmFlowStore  
  
    Dim FlowSession As SmartFlow.SmFlowSession
```

```
Dim FlowQueueItem As SmartFlow.SmFlowQueueItem

Dim ActiveProcess As SmartFlow.SmActiveProcess

Dim Response As SmartFlow.SmResponse


' Get FlowStore object - Sm Flow Service
Set FlowStore = SmSession.GetService("SmartFlow.SmFlowStore")

' Get current user flow session
Set FlowSession = FlowStore.FlowSession

' Check if exist process in Inbox of smartbox
If FlowSession.InboxProcesses.Count > 0 Then

    ' Retrieve first process from user inbox
    Set FlowQueueItem = FlowSession.InboxProcesses.Item(0)

    ' Check if can capture process - may be captured by another user
    If FlowQueueItem.CanCapture Or FlowQueueItem.PendingStatus =
fisCapture Then

        ' Capture process
        FlowQueueItem.Capture

        ' Get active process
        Set ActiveProcess = FlowQueueItem.ActiveProcess

        ' Check for tasks that process has not performed
        If FlowQueueItem.AppropriateTasks.Count > 0 Then

            ' Select all linked object for tasks
            FlowQueueItem.SelectAllObjects

            ' Select all tasks not performed
            FlowQueueItem.SelectedTasks = FlowQueueItem.AppropriateTasks

            ' Execute all selected tasks
            ActiveProcess.ExecuteSelectedTasks

        End If

    End If

End If
```

```
' Get first not reject response for current item

Set Response = FlowQueueItem.Node.GetNotRejectResponses(0)

' Check if process can be sent

If FlowQueueItem.CheckSendPossibility(Response) Then

    ' Send active process with comment

    ActiveProcess.Send Response, "Sent", Date

End If

End If

End Sub
```

ActiveProcess Task: Sending the FlowProcess

Use the **Send** method to send the FlowProcess to the next nodes according to the specified response. You can add a comment and specify the date.

ActiveProcess Task: Sending an E-Mail Ahead

Use **SendEmailToFollowingNodes** to display an empty mail item screen with the To: field filled in with the Users on the following Nodes. You fill in the message and send it.

FlowSentProcesses Task: Working with the FlowSentProcesses Collection

The FlowSentProcesses collection includes all FlowSentProcess objects of a given type for the current user.

To get a FlowSentProcess from the FlowSentProcesses collection by index, use the **Item** property.

To refresh the FlowSentProcesses collection, use the **Refresh** method.

To check if the current FlowSentProcesses collection contains completed FlowSentProcess objects, use the **Completed** property.

To determine the type of FlowSentProcess objects in the collection, use the **SentType** property. The results can be:

SentType	Description	Software Constant
Sent	Collection of SentProcess items related to this u which are not yet completed Collection of SentProcess items related to this u which are completed	qtSent
Deleted	A Process can arrive at this queue after being deleted at SentProcessesNotCompleted or SentProcessesCompleted	qtDeleted

SmWorkflowView Object

Description

The **SmWorkflowView** object represents a user interface relative to a FlowProcess, a User and a Node. It can be used to view FlowQueueItems and also SentProcess items.

The **SmWorkflowInitiateView** object is similar to the **SmWorkflowView** object but in this view you can initiate a process.

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to a WorkflowView and WorkflowInitiateView.

WorkflowView Task: Working with the WorkflowView Object

Use the **Process** property to set or get the FlowProcess that is displayed on the user interface.

Example

```
Sub DisplayProcesses(SmSession As SmApplic.SmSession)

    Dim FlowStore As SmartFlow.SmFlowStore

    Dim WorkFlowView As SmartFlow.ISmWorkflowView

    ' Get FlowStore object - Sm Flow Service
    Set FlowStore = SmSession.GetService("SmartFlow.SmFlowStore")

    ' Create new view
    Set WorkFlowView = FlowStore.NewWorkflowView

    ' Check if has processes in inbox
    If FlowStore.FlowSession.InboxProcesses.Count > 0 Then

        ' Assign process to view
```

```

        Set WorkflowView.Process =
FlowStore.FlowSession.InboxProcesses(0).FlowProcess

        ' Show process

        WorkflowView.Show

    Else

        MsgBox "No processes in your inbox"

    End If

End Sub

```

Use the **User** property to set or get the user whose FlowProcess is displayed on the user interface.

Use the **Node** property to set or get the node at which the information is to be displayed.

Use the **ReadOnly** property only for FlowQueueItem objects at the Inbox to prevent actions on the user interface.

Use the **BoxType** property to determine which type of box the WorkflowView is currently displaying. The Box types are described in the following table.

BoxType	Description	Software Constant
Inbox	A display of all process that have arrived for this user.	bxtInbox
Sent	A display of all processes that have been sent by this user.	bxtSent
Deleted	A display of all processes that have been deleted by this user.	bxtDeleted
Completed	A display of all processes that have been completed.	bxtCompleted

Use the **Show** method to cause the object to be displayed.

Use the **Style** property to determine the window style of the user interface, as described in the following table.

Style	Description	Software Constant
Normal	Defines an independent window	swsNormal
MDIChild	Defines a child window inside a parent window	swsMDIChild

WorkflowInitiateView Task: Creating a New FlowProcess and Viewing it.

This section describes the methods and properties used to create a new process under the user interface WorkflowInitiateView.

Use the **ProcessClassId** property to select a process ClassId for the FlowProcess you want to create. Alternately, use the **ProcessClass** property to create the FlowProcess class.

Use the **AttachedObjects** property to specify the objects you want to be attached to the newly created process.

Use the **Show** method to create the new process, to attach the default flowchart, attach the objects to the FlowProcess, and to display them.

Use the **Style** property to determine the window style of the user interface (see WorkflowView for an explanation).

SmFlowStore Object

Description

The **SmFlowStore** object provides access to the functionality of the **SmarTeam - Workflow** library. Using the SmFlowStore object, you can perform the following:

- Access the associated FlowSession object using the FlowSession property
- Access the associated MessageStore object using the MessageStore property
- Access collection objects such as FlowCharts, Responses, ProcessAssignments
- Create new instances of related objects, such as FlowChart, FlowProcess and others.
- Retrieve FlowProcess objects and other objects from the database.

Object Diagram

The following diagram shows the major properties of the SmFlowStore object:

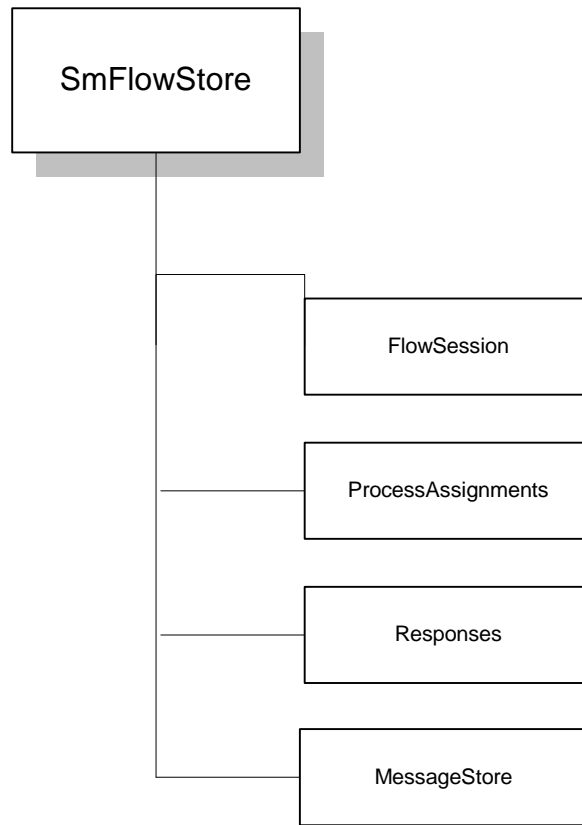


Figure 8-4 *FlowStore* Object Access

Obtaining the SmFlowStore Objects

For a stand-alone application:

```
Set FlowStore = Nothing
```

```
Set FlowStore = SmSession.GetService("SmartFlow.SmFlowStore")
```

Scripts associated with an event or a task are provided with a reference to an SmFlowProcess object. You can use this reference to obtain the SmFlowStore object.

```
Set FlowStore = ActiveProcess.FlowStore
```

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to a FlowStore.

FlowStore Task: Create a New FlowProcess

The **NewFlowProcess** method creates a new SmFlowProcess object but does not save it in the Database.

The **InitiateNewProcess** method creates a new SmFlowProcess object, saves it in the Database, and attaches a dummy default Flowchart to it.

Example

```
ProcessClassId = SmSession.MetaInfo.SmClassByName("General Process").ClassId
```

```
Set ChildProcess = FlowStore.InitiateNewProcess(ProcessClassId)
```

FlowStore Task: Create New Workflow View

Use the **NewWorkflowView** method to create a new WorkflowView object.

Use the **NewWorkflowInitiateView** method to create a new WorkflowView object in which you can initiate a FlowProcess.

```
Set WorkflowInitiateView = FlowStore.NewWorkflowInitiateView
```

FlowStore Task: Verify SmarTeam - Workflow Server

Use the **FlowServerInUse** property to verify if the system works with the **SmarTeam - Workflow Server**.

Example

```
Set FlowStore = SmSession.GetService("SmartFlow.SmFlowStore")
```

```
If FlowStore.FlowServerInUse Then
```

```
    MsgBox "Flow system is working with the SmartFlow Server"
```

```
End If
```

Overview of the SmartMessage Library Objects

This section presents an overview of the main SmartMessage objects including a description of the associated objects that are useful for the programmer.

- SmMessageSession
- SmMessageQueue
- SmMessages
- SmMessage
- SmExternalMessage
- SmMessageStore

SmMessageSession Object

Description

The **SmMessageSession** object represents the message context of a single user. It maintains a collection of that user's Inbox messages, deleted messages, draft messages and sent messages.

Object Diagram

The object diagram of **SmMessageSession** is shown below:

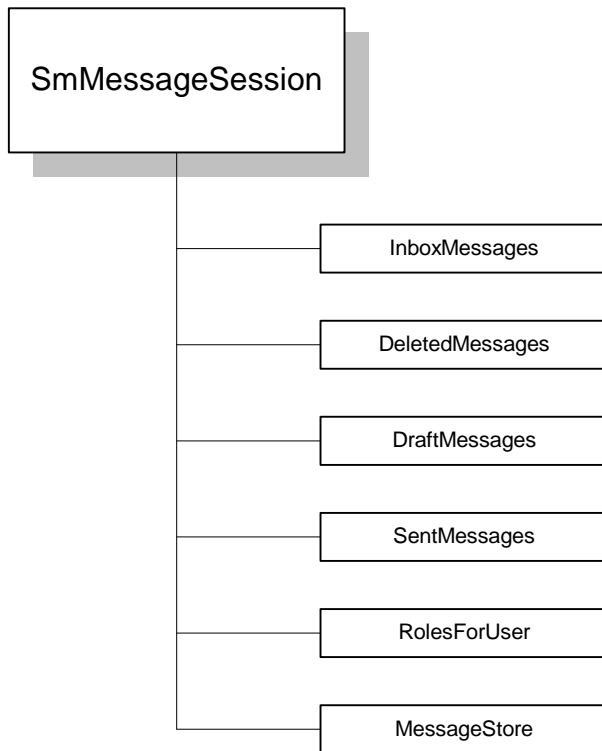


Figure 8-5 MessageSession Object Diagram

Obtaining a SmMessageSession Object

```
Set MessageSession = SmMessageStore.MessageSession
```

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to an SmMessageSession object and its components.

MessageSession Task: Getting the User's Message Context

Use the **User** property to get the User associated with the current MessageSession.

Use the **InboxMessages** property to get the ISmMessageQueue collection of messages in the InBox queue

Use the **SentMessages** property to get the ISmMessageQueue collection of messages in the Sent Messages queue

Use the **DeletedMessages** property to get the ISmMessageQueue collection of messages in the Deleted Messages queue

Use the **DraftMessages** property to get the ISmMessageQueue collection of messages in the Draft Messages queue.

SmMessageQueue Object

Description

The **SmMessageQueue** object represents a collection of the user's messages, including Deleted, Draft, In, and Sent messages.

The **SmMessage** object represents an individual message for a user that is sent within the **SmarTeam – Editor** application.

The **SmExternalMessage** represents an individual message that is handled by an external mail program such as Microsoft Outlook.

Object Diagram

The object diagram of SmMessageQueue is shown below:

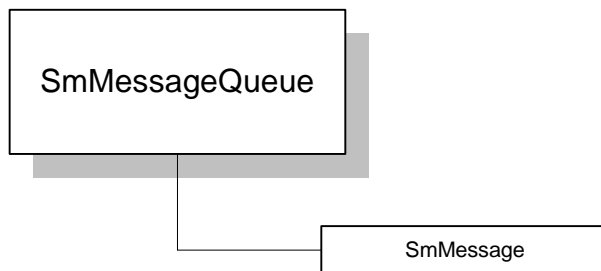


Figure 8-6 MessageQueue Object Diagram

Obtaining a SmMessageQueue Object

To get a MessageQueue object:

```
Set InboxQueue = SmMessageSession.InboxMessages
```

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to an SmMessageQueue object and its components.

MessageQueue Task: Working with the MessageQueue Collection

Use the **MessageQueueType** property to determine the type of queue that is represented by the current MessageQueue. The queue types are described in the following table.

QueueType	Description	Software Constant
In	Collection of InBox Queue items related to messages of user	qtIn
Sent	Collection of SentMessage items related to this user	qtSent
Deleted	Collection of DeletedMessage items related to user.	qtDeleted
Draft	Collection of DraftMessage items related to this user.	qtDraft

To get an individual Message object from the MessageQueue collection by index, use the **Item** property.

To get the index of a specified Message object in the collection, use the **IndexOf** method.

Use the **Refresh** method to refresh the MessageQueue to include new Message objects.

Message Task: Getting the Elements of a Message Object

The following properties represent the components of a message object:

AttachedObjects represents an ISmMultiObjects collection of **SmarTeam** objects attached to the message.

Body contains the body of the message

CCAsString represents the list of CC recipients, separated by semi-colons

CCList represents the list of CC recipients, as a collection of `IsmCompositeObjects`.

CreationDate represents the creation date of the message

Deleted represents a boolean value for the deletion status of an object. It is True if the object is marked as deleted.

DeletedStatus is the deletion status of the message. It can have the values:

- `DsNotdeleted`
- `DsMarkedAsDeleted`
- `dsDeleted`

From represents the composer of the message, as an `ISmCompositeObject`

OriginalMessage represents the previous message in the message thread of this message

ReceiptDate represents the date the message was received.

Use the **FromAsString** property to get the From: field of the message as a string.

Use the **ToAsString** property to get the To: field of the message as a string.
Use the **Importance** property to get the importance of the message. The importance property can have the following values:

Importance	Description	Software Constant
High	The Message is very important	<code>iiHigh</code>
Low	The Message has low importance	<code>iiLow</code>
Normal	The Message has normal importance	<code>iiNormal</code>

Use the **Subject** property to get the Subject of message as a string

Use the **MessageType** property to get the message type. The message types are listed in the following table:

Message Type	Description	Software Constant
Message	Regular message	mtMessage
Process	Internal use only	mtProcess
Decline	SmarTeam - Workflow sends this message to the Supervisor when a process is declined by an Executor.	mtDecline
DeclineAccept	Not in use	mtDeclineAccept
Capture	If the flag is set, SmarTeam - Workflow sends this message to the Supervisor when a process is captured by an Executor.	mtCapture
Consult	Not in use	mtConsult
Reject	Not in use	mtReject

Message Task:
Adding Items to a Message Object

Use the **AttachObject** method to attach the specified object to the message.

Use the **AddRecipient** method to add a recipient to the message according to the specified role. The message roles are described in the following table.

Message Role	Description	Software Constant
From	The composer of the message	mrFrom
To	The receiver of the message	mrTo
CC	The CC receiver of the message	mrCC

ExternalMessage Task:
Getting the Elements of a ExternalMessage Object

The following properties represent the components of an external message object:

- AttachedObjects
- Body
- CCAsString
- CCList
- CreationDate
- Deleted
- DeletedStatus
- From
- Importance
- OriginalMessage
- ReceiptDate

Use the **FromAsString** property to get the From field of the message as a string.

Use the **ToAsString** property to get the To field of the message as a string.

Use the **CCAsString** property to get the CC field of the message as a string.

Use the **Importance** property to get the importance of the message.

Use the **Subject** property to get the Subject of message as a string

Use the **Body** property to get the Text of the message

Use the **MessageType** property to get the message type. The message types are listed in the following table:

**ExternalMessage Task:
Adding Items to an External Message Object**

Use the **AttachObject** method to attach the specified object to the message.

Use the **AddRecipient** method to add a recipient to the message according to the specified role. The message roles are described in the following table.

Message Role	Description	Software Constant
From	The composer of the message	mrFrom
To	The receiver of the message	mrTo
CC	The CC receiver of the message	mrCC

SmMessageStore Object

Description

The **SmMessageStore** object is the root object for the **SmartMessage** library. It provides access to the other objects in the **SmartMessage** Object Model, and enables creation of new objects such as messages and message queues.

Object Diagram

The object diagram of SmMessageStore is shown below:

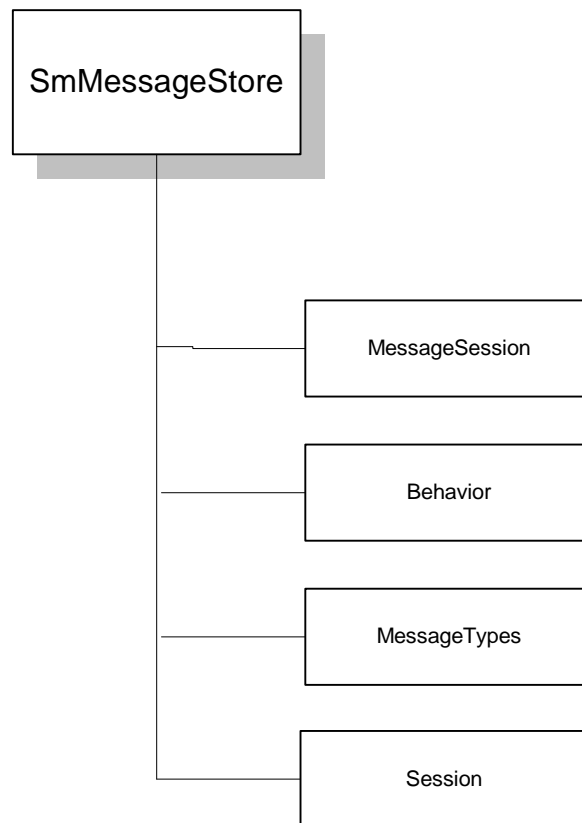


Figure 8-7 MessageStore Object Diagram

Obtaining a SmMessageStore Object

To obtain a MessageStore object:

```
Set MessageStore = SmSession.GetService( "SmartMessages.SmMessageStore" )
```

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to an SmMessageStore object and its components.

MessageStore Task: Creating New Message Objects

The **MessageSession** property is a reference to the MessageSession object.

Use the **NewSmartMessage** method to create a new SmartMessage object.

Use the **NewExternalMessage** method to create a new ExternalMessage object.

Use the **OpenSmartMessage** method to open a SmartMessage according to its objectID.

Using the SmarTeam - Workflow Library

This section explains how to write code that operates in conjunction with **SmarTeam - Workflow**.

You can use the **SmarTeam - Workflow** Library to write a **SmarTeam - Workflow** application or you can use it to write run-time script.

You can write a **SmarTeam - Workflow** application which initiates **SmarTeam – Editor** to perform a task. For example, you could write an application to create and send a process.

Run-time script is designed to perform tasks relating to a currently executing **SmarTeam - Workflow** session. You might use it, for example, to automatically send an e-mail or a message when a process is sent out from a specific node.

Transactions in the SmarTeam - Workflow Library

The following **SmarTeam - Workflow** methods manage transactions by themselves. Therefore, they cannot be called when a transaction is open. If a transaction is open, a call to one of these methods raises an exception.

ISmActiveProcess.Send

ISmServerQueueItem.Send (is used in **SmarTeam - Workflow** server)

ISmFlowStore.InitiateNewProcess

ISmFlowProcess.InitiateProcess

ISmFlowProcess.FullFlowchartCopy

ISmFlowProcess.SaveChangedFlowChart

Writing SmarTeam - Workflow Applications

Creating a FlowProcess

This section presents an example of a stand-alone application that creates a new General Process FlowProcess and sends it from the Start Node

```
Sub Main()
```

```
Dim SmEngine As SmApplic.SmEngine '(lib.class)

Dim SmSession As SmApplic.SmSession

Dim CommonGui As SmGUISrv.SmCommonGUI

Dim ObjectToSend As SmApplic.ISmObject

Dim FlowStore As SmartFlow.SmFlowStore

Dim FlowProcess As SmartFlow.SmFlowProcess

Dim ProcessClassId As Integer

Dim FlowSession As SmartFlow.SmFlowSession

Dim Node As SmartFlow.SmNode

Dim Response As SmartFlow.SmResponse

Dim Comment As String


' uses only visual basic - you don't need to run SmarTeam first
' create SmarTeam engine object
Set SmEngine = CreateObject("SmApplic.SmEngine")

' initialize object
SmEngine.Init "SmTeam32"

' creates a SmarTeam session for user
Set SmSession = SmEngine.CreateSession("Test Session", "Smart32")

' get a database from the first in the database collection
Set Database = SmEngine.Databases(0)

' connect SmSession to Database
SmSession.OpenDatabaseConnection Database.Alias, Database.Password, True


' create CommonGui object for SmarTeam views
Set CommonGui = SmSession.GetService("SmGUISrv.SmCommonGUI")

' method opens the login dialog box
CommonGui.Dialogs.ExecuteLogin
```

```
' exit if user didn't log in

If Not SmSession.UserLoggedIn Then

    Exit Sub

End If

' get ClassId of process type General Process

ProcessClassId = SmSession.MetaInfo.SmClassByName("General
Process").ClassId

' creates FlowStore object for flow operations

Set FlowStore = SmSession.GetService("SmartFlow.SmFlowStore")

' create FlowSession object for currently logged-in user

Set FlowSession = FlowStore.FlowSession

' gets first object from user-selected list -- SmObject

Set ObjectToSend =
CommonGui.Dialogs.ExecuteSelectFromQueryResult(0).Item(0)

' creates FlowProcess object of type General Process

Set FlowProcess = FlowStore.InitiateNewProcess(ProcessClassId)

' establish a complex link between ObjectToSend and the FlowProcess

Set AttachedObject = FlowProcess.LinkObject(ObjectToSend, Nothing)

Comment = "Auto sended process"

' creates node object representing Start Node

Set Node = FlowProcess.Flowchart.StartNode

' From StartNode only accept connectors exist

' create a response object from the response of the first outconnector

Set Response = Node.OutConnectors.Item(0).Response

' send FlowProcess from start node on all connectors with that Response

FlowProcess.InitiateProcess FlowSession, Response, Comment, Date

End Sub
```

Writing Run-Time Scripts

You can cause **SmarTeam - Workflow** to execute your script in run-time by attaching the script to “hooks” that are provided for this purpose in the Flow Chart Designer. You attach the script by using the appropriate dialog boxes when you design a flow chart.

There are two types of hooks provided: those associated with a task and those associated with an event.

When the Flowchart is assigned to a FlowProcess, the script is assigned as well. Then as the FlowProcess executes and events occur and tasks are performed, the associated script is called and runs automatically.

Task-Driven Scripts

A task-driven script is associated with a task object within a flow chart when the flow chart is designed.

You can specify that the script be executed.

You can attach automatic script to the following types of events:

Event Type	Description
On Capture	This event occurs when a user captures a process. A user can capture a process either explicitly or implicitly. A process is captured explicitly through the InBox window. A process is captured implicitly when the SmarTeam user performs an act that requires capture privileges.
On Respond	This event occurs when the user presses Accept or Reject to send a process. It is similar to the Before Send event.

Script Options

The following options are provided for a task:

Task Option	Description
Required task	The task must be performed
Perform task per object	The task is performed on all objects attached to the process.
Perform task automatically	Select whether the script is executed when the user performs a task on the Process window, or when the events On Capture or On Respond occur.
--On Capture	
--On Respond	
Class	The script is executed for the objects in the specified class.

Timing for Task-Driven Scripts

Figure 8-8 shows when a task-driven script is executed relative to a **SmarTeam - Workflow** user action. For example, when a **SmarTeam** user performs a capture process action, the On-Capture event occurs. If an automatic task that has a script is attached to that event, the script will be executed.

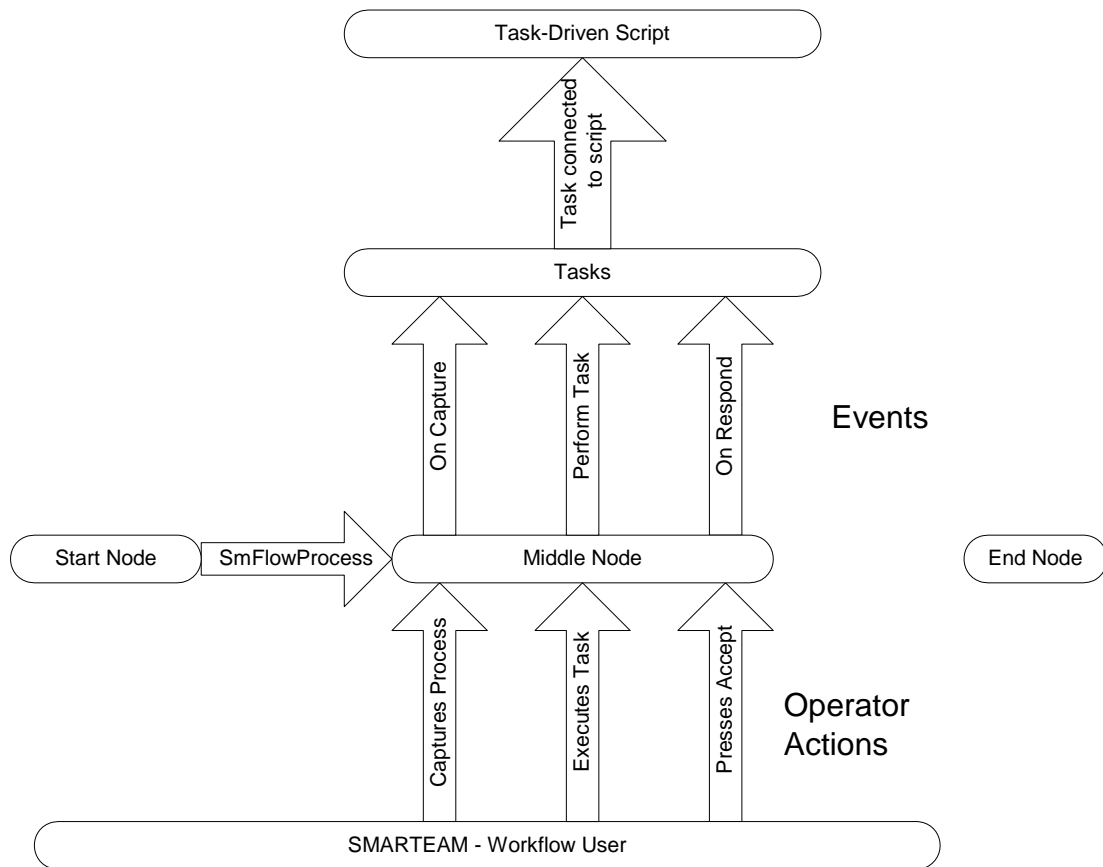


Figure 8-8 Time-Line Chart for Task-Driven Scripts

Script Format

A task-driven script has the following format:

```
Sub Task-Driven (ActiveProcess As ISmActiveProcess
                Task As ISmTask
                MObjects As ISmMultiObjects)
```

OnCapture Example

This section presents an example of a task-driven script. The script sends an e-mail to all users on previous Nodes notifying them that the current user captured a process that they sent. The OnCapture event causes this script to be executed.

```
Sub OnCapture(
    ActiveProcess As Object, 'the active process you captured
    Task As Object,          'the task that activated this script
    MultiObjects As Object   'objects attached to Active Process
)

Dim FromNodes As SmartFlow.SmNodes

Dim CurrentNode As SmartFlow.SmNode

Dim UserLogin As String

Dim Users As Object

Dim CurrentUser As SmApplic.ISmObject

Dim enumMailItem As Integer

Dim Mail As Object

Dim MailServer As Object

Dim i As Long

Dim j As Long

' get the current UserLogin from the current user's data
UserLogin = ActiveProcess.Session.UserMetaInfo.UserLogin

' get the current node

Set CurrentNode = ActiveProcess.CurrentNode
```

```
' get collection of nodes that sent QueueItem to the current node
Set FromNodes = ActiveProcess.QueueItem.FromNodes
' create Outlook mailserver object(assuming Outlook is installed)
Set MailServer = CreateObject("Outlook.Application")
' set type of Outlook mail item to be "new message"
enumMailItem = 0
' create mail object
Set Mail = MailServer.CreateItem(enumMailItem)
' loop on FromNodes
For i=0 To FromNodes.Count - 1
    ' retrieve collection of users on all From Nodes
    Set Users = FromNodes(i).Users.GetUsers
    ' loop on users of a From Node
    For j=0 To Users.Count - 1
        ' create duplicate User object to get all attributes for that
        ' user (can't do that when the User in a collection)
        Set CurrentUser = Users(j).Clone
        ' get info from data base
        CurrentUser.Retrieve
        ' add the User email address from the database data record
        ' to the mail recipient collection
        Mail.Recipients.Add CurrentUser.Data.ValueAsString("USER_EMAIL")
    Next
Next
' Fill in mail text
Mail.Subject = "Capture process"
Mail.Body = "Process " + ActiveProcess.FlowProcess.Name + " was captured by "
+ UserLogin + " at node " + CurrentNode.Name
Mail.Send
End Sub
```

Event-Driven Scripts

An event-driven script is associated with an event object within a flow chart. When the event occurs, the script is executed automatically.

You can attach script to the following types of events:

Event Type	Description
On Receive	The On Receive event occurs when the SmFlowQueueItem corresponding to a SmFlowProcess enters a user's InBoxProcesses queue.
On Open	The On Open event occurs each time a WorkFlow screen corresponding to a SmFlowQueueItem is opened.
Before Send	The Before Send event occurs immediately before the process sent on to the next node.
Before Send Accept	Similar to Before Send described above Typical scripts for this event might be:
Before Send Reject	The Before Send Reject event occurs immediately before the process is rejected to a previous node.
After Send	The After Send event occurs after the process is sent to the next node.
After Send Accept	The After Send event occurs after the process is sent to the next node following the user's accept response.
After Send Reject	The After Send event occurs after the process is sent to a previous node following a user's reject response.

Timing for Event-Driven Scripts

Figure 8-9 shows when a event-driven script is executed relative to a **SmarTeam - Workflow** user action. For example, when a **SmarTeam** user presses accept at the Start Node, the BeforeSend event occurs. If a script is attached to that event, the script will be executed.

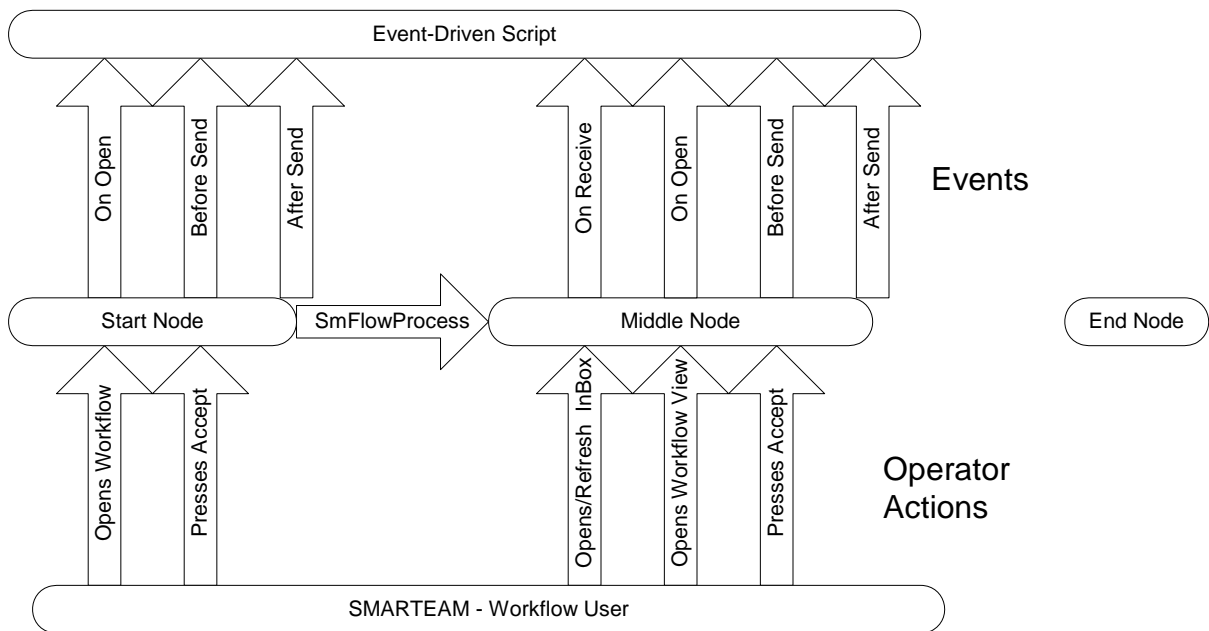


Figure 8-9 Time-Line Chart for Event-Driven Scripts

Script Format

A event-driven script has the following format:

```
Function BeforeSend(ActiveProcess As Object,  
                   Response As Object)As Integer
```

```
Function AfterSend(FlowSession As Object,  
                  FlowProcess As Object,  
                  Node As Object,  
                  Response As Object) As Integer
```

```
Function AfterSendReject(FlowSession As Object,  
                        FlowProcess As Object,  
                        Node As Object,  
                        Response As Object) As Integer  
  
Function AfterSendAccept(FlowSession As Object,  
                        FlowProcess As Object,  
                        Node As Object,  
                        Response As Object) As Integer
```

BeforeSend Example

The following sample script was written to be activated by a BeforeSend event.

The script CreateProcessLog creates a process log document and attaches it to a process and to a selected project. The log can be viewed by all users through the project window but it is not visible on the process view.

This script creates the log document and puts in the initial entry. To get a full process log for all stages of the process, a similar script, which adds information to the log, must be put in every subsequent Node.

```
Function CreateProcessLog(  
    ActiveProcess As Object, ' the Active Process you sent  
    Response As Object       ' the Response on which you sent it  
    ) As Integer  
  
    Dim SmSession As SmApplic.SmSession  
    Dim FlowProcess As SmartFlow.SmFlowSession  
    Dim Document As SmApplic.ISmObject  
    Dim NewTempDocument As SmApplic.ISmObject  
    Dim Project As SmApplic.ISmObject  
    Dim DocumentClassId As Integer  
    Dim GUIService As SmGUISrv.SmCommonGUI  
    Dim View As SMGUISrv.ISmView  
    Dim UserBehavior As SmApplic.ISmBehavior  
    Dim Link As SmApplic.ISmObject
```

```
Dim LinkAttributes As Object

Dim LinkClassId As Integer

Dim Task As Object

Dim SmSessionUtil As SmUtil.SmSessionUtil

Dim CurrentMask As String

Dim NextMask As String

Dim ObjectId As Long


' get flowprocess

Set FlowProcess = ActiveProcess.FlowProcess

' get SmSession

Set SmSession = ActiveProcess.Session

' get service for life cycle operation

Set SmSessionUtil=SmSession.GetService("SmUtil.SmSessionUtil")

' get class ID for class Document

DocumentClassId = SmSession.MetaInfo.SmClassByName("Document").ClassId

' create new Document object

Set NewTempDocument = SmSession.ObjectStore.NewObject(DocumentClassId)

' add Document empty attributes

NewTempDocument.AddAllAttributes

' fill in default attribute values

Set Document =NewTempDocument.FillDefaults

' insert description of Document object into Database

Document.Data.ValueAsString("CN_DESCRIPTION") = "Auto created for process
" & FlowProcess.Name

' get the current mask of the Document class primary identifier attribute
by its name CN_ID. Current mask is the mask of the most recently created
object of class Document, for example, "DOC-002".
```



```
CurrentMask=SmSessionUtil.RetrieveStartMaskValue
(Document.Attributes.ItemByName("CN_ID"))

' allocate a new primary identifier mask after the CurrentMask

NextMask = SmSessionUtil.RetrieveNextMask
(Document.Attributes.ItemByName("CN_ID"),CurrentMask)

' insert it as the Document object primary identifier attribute

Document.Data.ValueAsString("CN_ID") = NextMask


' alternative:

' If you want to use a log file which existed prior to executing this
script then add here a command to open a window to select the file
path\filename and:

' put the file name in the Document attribute " FILE_NAME "

' Document.Data.ValueAsString("FILE_NAME") = "filename"

' put the path in Document attribute "Directory"

' Document.Data.ValueAsString("DIRECTORY") = "path"


' create new file with name PrimaryIdentifier.txt and path C:\
Open "C:\" & NextMask & ".txt" For Output As #1

' write name of file creator

Print #1,"Created by user " & SmSession.UserMetaInfo.UserLogin

' write data and time

Print #1,"Date: " & Date$() & " Time: " & Time$()

' write name of process

Print #1,"For flow process " & FlowProcess.Name

' and name of Node

Print #1,"At node " & ActiveProcess.CurrentNode.Name

' response on send

Print #1,"On response " & Response.Name
```

```
Close #1

' put name of newly created file in Document attribute "FILE_NAME"
Document.Data.ValueAsString("FILE_NAME") = NextMask & ".txt"

' put path in Document attribute "DIRECTORY"
Document.Data.ValueAsString("DIRECTORY") = "C:\\"

' choose project to which the new Document is to be attached

' get GUI service
Set GUIService = SmSession.GetService("SmGUISrv.SmCommonGUI")

' show project view
Set View = GUIService.Views.NewViewByType(vwtMainClassTree)
View.ViewTitle = "Select project to link"

' open dialog box and let user choose project to link Document
View.SmViewWindow.ShowModal

' get the first one he selected
Set Project = View.Selected.Objects(0)

' use behavior that doesn't require confirmation
Set UserBehavior = SmSession.ObjectStore.DefaultBehavior.Clone

' set automatic confirmation - no prompt will be used
UserBehavior.ConfirmOperations = coYesToAll

' insert new document to data base
Document.InsertEx UserBehavior

' can't use zero directly as parameter, only by reference
LinkClassId = 0

' link Document to project

Set Link = SmSession.ObjectStore.NewOneLevelLink
(LinkClassId, Project.ClassId, Project.ObjectId, Document.ClassId, Document.ObjectId)

' insert link to database. the doc will appear linked to the project
Link.InsertEx UserBehavior
```

```
' only by reference

Set Task = Nothing

' check-in newly created document object to vault

ObjectId = SmSessionUtil.CheckIn(Document,Task,True)

Set LinkAttributes = Nothing

' secondary link document to process; doesn't appear on process view

Set Link = FlowProcess.LinkObjectAsSecondary(Document,LinkAttributes)

' alternative:

' primary link document to process; appears on process view

' Set Link = FlowProcess.LinkObject(Document,LinkAttributes)

End Function
```

BeforeSendAccept Example

The following sample script was written to be activated by a BeforeSendAccept event.

The script SendNewProcess creates a new flow process and assigns new users to nodes. It attaches objects from an existing process to the new process and sends the new process to selected users including both existing users and the new users.

```
Function SendNewProcess(

    ActiveProcess As Object, 'the active process you sent

    Response As Object      ' the Response on which you sent

    ) As Integer

    Dim CommonGUI As SmGUIsrv.SmCommonGUI

    Dim ViewedCompositeObjects As SmApplic.ISmCompositeObjects

    Dim Query As SmApplic.ISmSimpleQuery

    Dim SelectUserView As SmGUIsrv.ISmView

    Dim UserClassId As Integer

    Dim SmSession As SmApplic.SmSession
```

```
Dim DestinationNode As SmartFlow.SmNode

Dim Users As SmApplic.ISmObjects

Dim FlowStore As SmartFlow.SmFlowStore

Dim FlowSession As SmartFlow.SmFlowSession

Dim ChildProcess As SmartFlow.SmFlowProcess

Dim LinkedObjects As SmApplic.ISmMultiObjects

Dim SubLinkedObjects As SmApplic.ISmObjects

Dim CompLink As SmApplic.ISmObject

Dim Node As SmartFlow.SmNode

Dim OutNodes As SmartFlow.SmNodes

Dim ChildProcessResponse As SmartFlow.SmResponse

Dim ObjectToSend As SmApplic.ISmObject

Dim ProcessClassId As Integer

Dim Comment As String

Dim Param As Object

Dim I As Long

Dim J As Long


' get service object FlowStore
Set FlowStore = ActiveProcess.FlowStore

' get current user's FlowSession
Set FlowSession = ActiveProcess.FlowSession

' get the SmartTeam session
Set SmSession = ActiveProcess.Session

' create a new process

' get classid for General Process class

ProcessClassId = SmSession.MetaInfo.SmClassByName
("General Process").ClassId
```

```
' create and initiate new process, attach default Flowchart
Set ChildProcess = FlowStore.InitiateNewProcess(ProcessClassId)

' get all objects linked to Active Process without their links
Set LinkedObjects = ActiveProcess.FlowProcess.ObjectsData

' LinkedObjects has type IsmMultiObjects:
' including super classes Documents, Items, Users
For I = 0 To LinkedObjects.Count - 1
    ' create object for one superclass
    Set SubLinkedObjects = LinkedObjects(I)

    ' loop on that superclass collection
    For J = 0 To SubLinkedObjects.Count - 1
        ' for each member of collection
        Set ObjectToSend = SubLinkedObjects(J).Clone

        ' get all properties
        ObjectToSend.Retrieve

        ' link the object to the new process
        ' no link parameters, use defaults
        Set Param = Nothing

        Set CompLink = ChildProcess.LinkObject(ObjectToSend, Param)
    Next
Next

' get all SmarTeam users from Database
' create a new query object
Set Query = SmSession.ObjectStore.NewSimpleQuery

' define query to find all users defined in database
Query.SelectStatement = "Select * from USERS"

' run query
Query.Run
```

```
' transform from QueryResult (RecordList) to CompositeObjects for
viewing

Set ViewedCompositeObjects =
SmSession.ObjectStore.CompositeObjectsFromData(Query.QueryResult, true)

' get viewing service

Set CommonGUI = SmSession.GetService("SmGUISrv.SmCommonGUI")

Comment = "Auto created child process process"

' send the new process from the Start Node

Set Node = ChildProcess.Flowchart.StartNode

' send on accept connectors only

' select the first non-reject response for start node

Set ChildProcessResponse = Node.GetNotRejectResponses(0)

' get the collection of outgoing nodes for that response

Set OutNodes = Node.GetOutgoingNodes(ChildProcessResponse)

' loop over out nodes

For J = 0 To OutNodes.Count - 1
' create node object

    Set DestinationNode = OutNodes.Item(J)

    ' let the user choose new users to receive this process.

    ' check if new users can be added to the Destination Node

    If DestinationNode.RuntimeUsers Then

        ' make userview      window object

        Set SelectUserView = CommonGUI.Views.NewViewByType(vwtCustom)

        ' define view to include all users

        Set SelectUserView.DisplayObjects.CompositeObjects =
ViewedCompositeObjects

        ' set title of SelectUserView window

        SelectUserView.ViewTitle = "Select user for destination node "
& DestinationNode.Name
```

```
' open window and display all users for selection
SelectUserView.SmViewWindow.ShowModal

' create collection of all users selected
Set Users = SelectUserView.Selected.Objects

' loop over selected users
For I=0 To Users.Count - 1

    ' add selected users to destination node
    DestinationNode.Users.Add Users(I)

Next

' save destination mode with all added users
DestinationNode.Save

End If

Next

' send new process to the existing and new users
ChildProcess.InitiateProcess FlowSession, ChildProcessResponse, Comment,
Date

End Function
```

AfterSendAccept Example

The following sample script was written to be activated by an AfterSendAccept event.

The script SendMailAfterAccept sends e-mail to persons outside the system that the process has been sent.

```
Function SendMailAfterAccept(

    FlowSession As Object, 'flow session of user that sent process

    FlowProcess As Object, 'FlowProcess that was sent

    Node As Object,        'node from which it was sent

    Response As Object     'response on which process was sent

) As Integer
```

```
Dim DestinationNodes As SmartFlow.SmNodes

Dim Users As Object

Dim CurrentUser As SmApplic.ISmObject

Dim enumMailItem As Integer

Dim Mail As Object

Dim MailServer As Object

Dim i As Long

Dim j As Long

' create collection object of nodes to which process was sent
Set DestinationNodes = Node.GetOutgoingNodes(Response)

' create Outlook service main object
Set MailServer = CreateObject("Outlook.Application")
enumMailItem = 0

' mail item
Set Mail = MailServer.CreateItem(enumMailItem)

' loop on destination nodes
For i=0 To DestinationNodes.Count - 1
    ' create collection of users on a node
    Set Users = DestinationNodes(i).Users.GetUsers

    ' loop over users
    For j=0 To Users.Count - 1
        ' duplicate user to retrieve attributes
        Set CurrentUser = Users(j).Clone

        ' get attributes of user
        CurrentUser.Retrieve

        ' add recipient to mail list
```



```
        Mail.Recipients.Add
CurrentUser.Data.ValueAsString("USER_EMAIL")

        Next

    Next

    ' enter mail subject

    Mail.Subject = "Check your Smartbox"

    ' enter mail body

    Mail.Body =
"Process was sent to your SmartBox on response " + Response.Name

    ' send mail

    Mail.Send

End Function
```

9. SmarTeam CAD Interface Library

General Description

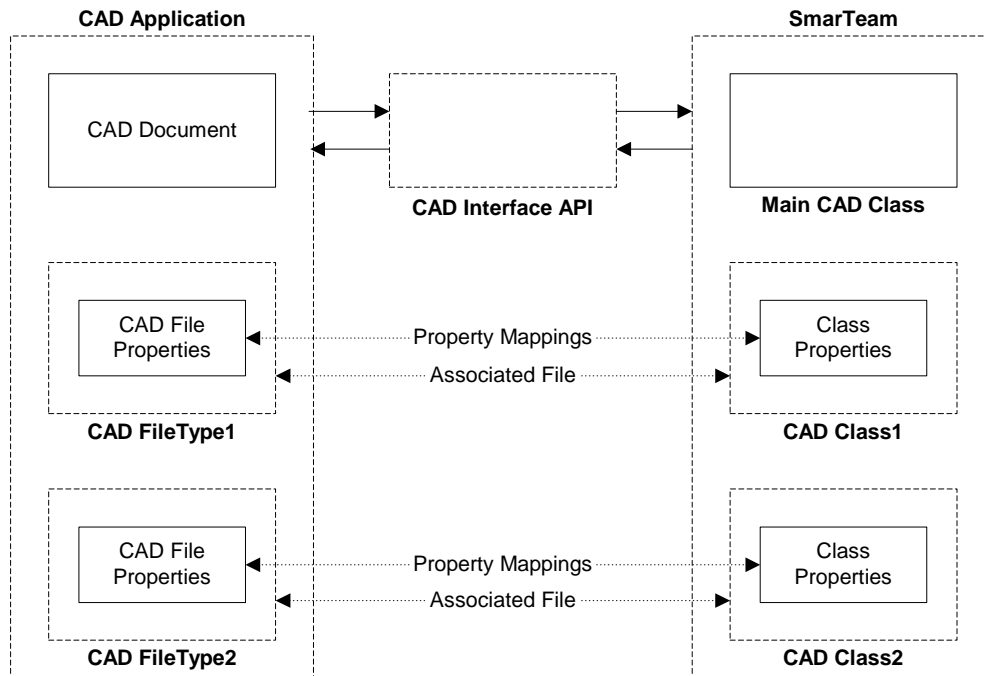
SmarTeam CAD Integration

The SmarTeam Integration Tool, by enabling you to map objects of an integrated CAD tool such as SolidWorks™ or Microsoft Word™ to objects in SmarTeam, lets you apply the power of SmarTeam to the CAD tool.

As shown in the following figure, the different CAD file types are mapped to SmarTeam classes and the CAD file property fields are mapped to the corresponding SmarTeam class attributes. For example, when a SolidWorks Part CAD file is mapped to the SmarTeam class SolidWorksPart, the Summary Information/Author property field of the CAD file can be mapped to the Object ID class attribute of the SolidWorksPart class. Now you can use that field in the CAD file for the SolidWorksPart User Object ID.

In addition, the CAD file is also linked to the corresponding SmarTeam class as an associated file; SmarTeam can manage the disposition of the CAD file in the SmarTeam vaults.

This chapter discusses the SmarTeam CAD Interface library, which can be used once the SmarTeam integration is in place and the mappings are established, for example, by using the SmarTeam Integration Tool.



Integration Data Model

The Integration Data Model uses the CLB and OLB mechanisms discussed in Chapter 5, section [Class Behaviors](#), to define possible object and link behaviors to support specific integrations. See that section for more information about Class Behaviors and SmDemo for examples.

The following terminology is used:

- **Integration Behavior** – An OLB, defined for the integration component objects, for example: *SolidWorks Part*, *Solid Edge Assembly*, and *CATIA Model*.
- **Integration Link Behavior** – A CLB imposed on a link class, which is used to distinguish between link classes, for example: *CATIA structure*, *CATIA contextual*, *SW Design in Context*.
- **Integration Composition** – A CLB composition, which determines the permissible link classes for two integration component objects.

Operation Dependency Rules – A set of restrictions on the life-cycle operations permissible for classes with given Integration Behavior or Integration Link Behavior.

SmarTeam CAD Interface

The SmarTeam CAD Interface library provides easy access to SmarTeam functionality from within an integrated CAD application, such as SolidWorks, or within a general-purpose application, such as Microsoft Word.

The functionality of the SmarTeam CAD Interface library includes:

- Improved file search capabilities, including searching for files by attributes other than the file name. The file search is carried out using SmarTeam search functions.
- The high-level functions of the CAD Interface library provide easy access to SmarTeam database information about edited documents and related documents.
- Improved file security; CAD files can be saved inside the SmarTeam vault immediately after creation or editing.
- Working directly with the CAD application files, saving file information in the SmarTeam database.

The SmarTeam CAD Interface library provides objects that enable you to:

- Save and update documents and compositions (trees) of the documents in the SmarTeam database.
- Retrieve document meta-information by file name from the SmarTeam database.
- Update various CAD blocks, for example, update the title block of a drawing with information obtained from SmarTeam.
- Perform life-cycle and other related operations.

Dependencies

The SmarTeam CAD Interface library has the following dependencies:

- SmarTeam Record List library.
- SmarTeam GUI Services library.
- SmarTeam Application library.
- SmarTeam Utilities library.
- SmarTeam Engine library.

Overview of Objects

The main object of the SmarTeam CAD Interface library is the SmCADInterface object.

SmCADInterface provides high-level functionality that enables you to maintain uniformity across all integrated applications, and to keep the integration updated when upgrades are implemented in SmarTeam and/or the applications.

It is possible to work with integrated applications using SmarTeam standard functions. However, for greater convenience and maintainability, it is highly recommended that you use the SmCADInterface object for all your SmarTeam integration requirements.

SmCADInterface Object

The SmCADInterface object provides the following functionality, including:

- Connecting to the SmarTeam database
- Updating the SmarTeam database
- Retrieving information from SmarTeam database
- GUI operations
- Lifecycle operations
- Improving Performance

Object Diagram

The object diagram of SmCADInterface is shown below:

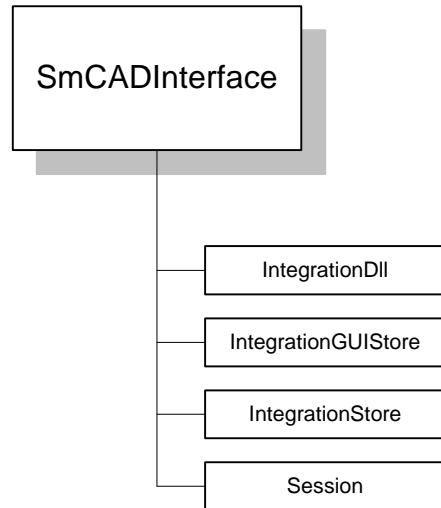


Figure 9-1 SmCADInterface Object Diagram

Obtaining the SmCADInterface Object

SmCADInterface is a service object. To obtain a reference to the object, use the following syntax:

```
Dim CADInterface as SmCADInterface  
  
Set CADInterface = Session.GetService ("SmCad.SmCADInterface")
```

SmCADInterface Properties

The SmCADInterface has the following properties:

IntegrationGUIStore	Gets the ISmIntegrationGUIStore object of the current session
IntegrationStore	Gets the ISmIntegrationStore object of the current session
Session	Returns an ISmSession object that represents the current session

The following objects can be accessed through the properties of **SmCADInterface**.

SmSession

The **SmSession** property enables you to access the associated **SmSession** object from **SmCADInterface**:

```
Dim SmSession As ISmSession  
Set SmSession = CADInterface.Session
```

SmIntegrationStore

The **SmIntegrationStore** property enables you to access the **SmIntegrationStore** object from **SmCADInterface**:

```
Dim SmIntegrationStore As ISmIntegrationStore  
Set SmIntegrationStore = CADInterface.SmIntegrationStore
```

SmIntegrationGUIStore

The **SmIntegrationGUIStore** enables you to retrieve the **SmIntegrationGUIStore** object from **SmCADInterface**:

```
Dim SmIntegrationGUIStore As ISmIntegrationGUIStore  
Set SmIntegrationGUIStore = CADInterface.SmIntegrationGUIStore
```

SmCADInterface Methods

The SmCADInterface has the following major methods, presented according to topic:

Database Connection	
Initialize	Initializes variables, creates database connection and login
UserLogin	Connects and login into the SmarTeam database
Terminate	Terminates the session and disconnects from database
Updating SmarTeam Database	
UpdateLinks	Updates links in the database, updating only those that have changed, and adds the new links
SaveObjectsAndLinks	Updates objects and links in the SmarTeam database, updating only those that have changed, and adds the new objects and links.
OdmaSave	Displays the Projects Manager dialog and saves the document in SmarTeam database
Save	Saves the document in SmarTeam database
Retrieving Information from SmarTeam Database	
FindObject	Retrieves Object Id and Class Id for the object (Part or Document) to which the input CAD document is associated.
FindObjects	Retrieves Object Id and Class Id for the objects (Part or Document) to which the input CAD documents are associated
GetLinkedObjects	Retrieves Objects that are linked with a specified link behavior the object (Part or Document) to which the input CAD document is associated and returns specified object attributes.
FindFile	Locates the full path of the CAD document associated with the specified SmarTeam object. In case of multiple files, the first found is returned.
GUI Operations	
Locate	Locates a document and shows its profile card
WhereUsed	Displays the document's Where Used list
Life-Cycle Operations	
Approve	Performs the Release operation
CheckIn	Performs the CheckIn operation
CheckOut	Performs the CheckOut operation
CheckOutForEdit	Performs the CheckOut and Edit operations
CopyFile	Performs a CopyFile operation
NewRelease	Performs the NewRelease operation
Obsolete	Performs the Obsolete operation

Performance	
BeginSaveOperation	Initializes global variables for the save operation
EndSaveOperation	Releases global variables after the save operation
GetChildrenWithCopies	Retrieves all the assembly's children from the local folders
Miscellaneous	
ApplUpdateProperties	Updates the document's properties before it is moved to the v – fires event OnLFCOperation
InvokeScriptWithList	Invokes a script with an input record list
SetMainWindowHandle	Sets the parent window of SmarTeam API windows
ShowSmarTeam	Activates the SmarTeam application
GetDefaultWorkFolder	Retrieves current work folder
GetTemporaryFolder	Retrieves current folder for temporary files
IsOperationAllowed	Checks if operation may be performed
IsLinkAllowed	Checks if two documents can be linked
TransferOwnership	Transfers document from one user to another in the collabora design environment

Using the File Description Record List

Many of the SmCADInterface methods use a record list of a certain format to identify documents that are associated with SmarTeam objects. The record list should include one or more records, with the following headers:

Header	Type	Size	Description
FILE_NAME	CHAR	256	Full file name, including directory and file name
TDM_COMPONENT_NAME	CHAR	256	Name of the component or configuration, r be empty
INTEGRATION_BEHAVIOR	CHAR	256	Integration behavior name as defined in th integration's data model.

Example

```
Dim FullPath As String

FileDescription.AddHeader "FILE_NAME", 256, TDMT_CHAR

FileDescription.AddHeader "INTEGRATION_BEHAVIOR", 256, TDMT_CHAR

FileDescription.AddHeader "TDM_COMPONENT_NAME", 256, TDMT_CHAR

FileDescription.Value("FILE_NAME", 0) = "c:\work\p01.sldprt"
```

```
FileDescription.Value("INTEGRATION_BEHAVIOR",0) = "TDM_SW_PART"
```

```
FileDescription.Value("TDM_COMPONENT_NAME",0) = "Default"
```

In addition to the attributes shown above, **FileDescription** can also include other attributes that are added to the specified document's profile card during Save/Update operations.

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to SmCADInterface.

CADInterface Task: Connecting to the SmarTeam Database

The first call to the **SmCADInterface** object must be to the method **Initialize**. This method creates the connection to the SmarTeam database and opens the login screen, as shown in this example:

```
Dim RetCode as Integer

Dim CADInterface as SmCADInterface

Set CADInterface = Session.GetService ("SmCad.SmCADInterface")

RetCode = CADInterface.Initialize("Microsoft Word")
```

To deactivate the integration, call the **Terminate** method. This method disconnects the user from the SmarTeam database:

```
CADInterface.Terminate()
```

CADInterface Task: Adding/Updating Document Descriptions

The **OdmaSave** method enables you to add or update a document's description inside the SmarTeam database, as shown in the following example:

```
Dim RetCode As Integer

Dim ShowSaveAsDialog as Integer

Dim ForceSaveAs as Integer

ShowSaveAsDialog = 1

ForceSaveAs = 0
```

```
RetCode = SmCADInterface.OdmaSave(FileDescription, ShowSaveAsDialog,  
ForceSaveAs)
```

CADInterface Task: Adding/Updating Document Links

The `SaveObjectsAndLinks` and `UpdateLinks` methods allow you to add and update SmarTeam links between one document object and other related document objects. The document's links are added, deleted or updated in the SmarTeam database to reflect the current links in the CAD integration.

Updating Links

When the SmarTeam objects corresponding to the CAD documents are related by links with specific behavior, you update them by calling the method `SaveObjectsAndLinks` or `UpdateLinks`. The difference between these methods is that `SaveObjectsAndLinks` updates the links and also saves the objects that were not yet saved in the SmarTeam database.

For example, consider a SolidWorks assembly A1 that has SolidWorks parts P1 and P2 as components. A1, P1, and P2 are represented by SmarTeam objects that are linked with links with behavior "Composed Of": $A1 \rightarrow P1$ and $A1 \rightarrow P2$.

Suppose that, in the CAD integration, the user deletes part P2 from the assembly A1 and adds part P3. In order to update the SmarTeam database accordingly, you call `UpdateLinks` with assembly A1 as a single record in the `FileDescription` parameter and the two components P1 and P3 as separate records in the `References` parameter and a single record containing `TDM_SW_COMPOSED_OF` record in the `LinkBehaviors` parameter. The method acts to delete SmarTeam link $A1 \rightarrow P2$, and to add a new SmarTeam link $A1 \rightarrow P3$.

Note: You must make a separate call to `UpdateLinks` for each hierarchical level in the assembly in which a change was made.

Note: If P1 has two instances (records) inside the `References` record list, then its link quantity value will be set to "2".

Note: If any of the documents referred to the input record lists do not yet exist as objects inside SmarTeam and the `SaveObjectsAndLinks` is called, then a new profile card will be created for them.

Example

The following example shows how to use the method `SaveObjectsAndLinks` to update a SolidWorks assembly in the SmarTeam database.

```
Dim RetCode As Integer

Dim FileDescription As ISmRecordList

Dim References As ISmRecordList

Dim LinkBehaviors As ISmRecordList

' Load the assembly information into the FileDescription record list
Set FileDescription = New SmRecList.SmRecordList
FileDescription.AddHeader "FILE_NAME",256,TDMT_CHAR
FileDescription.AddHeader "INTEGRATION_BEHAVIOR",256,TDMT_CHAR
FileDescription.AddHeader "TDM_COMPONENT_NAME",256,TDMT_CHAR

FileDescription.Value("FILE_NAME",0) = "c:\work\p01.sldASM"
FileDescription.Value("INTEGRATION_BEHAVIOR",0) = "TDM_SW_ASSEMBLY"
FileDescription.Value("TDM_COMPONENT_NAME",0) = "Default"

' Load each part and document in a separate record in the
' References record list
Set References = New SmRecList.SmRecordList
References.AddHeader "FILE_NAME",256,TDMT_CHAR
References.AddHeader "INTEGRATION_BEHAVIOR",256,TDMT_CHAR
References.AddHeader "TDM_COMPONENT_NAME",256,TDMT_CHAR
References.AddHeader "LINK_BEHAVIOR",256,TDMT_CHAR

References.Value("FILE_NAME",0) = "c:\work\p01.sldprt"
```

```
References.Value("INTEGRATION_BEHAVIOR",0) = "TDM_SW_PART"

References.Value("TDM_COMPONENT_NAME",0) = "Default"

References.Value("LINK_BEHAVIOR",0) = "TDM_SW_COMPOSEDOF"


References.Value("FILE_NAME",1) = "c:\work\table1.xls"

References.Value("INTEGRATION_BEHAVIOR",1) = "TDM_EXCEL_DOCUMENT"

References.Value("TDM_COMPONENT_NAME",1) = ""

References.Value("LINK_BEHAVIOR",1) = "TDM_SW_TABLE_LNK"


Set LinkBehaviors = New SmRecList.SmRecordList

' Load link behaviors specified in References parameter

LinkBehaviors.AddHeader "LINK_BEHAVIOR",256,TDMT_CHAR

LinkBehaviors.Value("LINK_BEHAVIOR",0) = "TDM_SW_COMPOSEDOF"

LinkBehaviors.Value("LINK_BEHAVIOR",1) = "TDM_SW_TABLE_LNK"


RetCode =SmCADInterface.SaveObjectsAndLinks (FileDescription,
References,LinkBehaviors)
```

CADInterface Task: Retrieving Document Information

The **FindObject** method enables you to retrieve a document's class ID and object ID.

The document is searched inside the set of classes, which are defined for its FILE_TYPE. The database query is based on the full path from the FILE_NAME header, which is used for the file name and directory fields.

If the "Additional Identifier" attribute is set inside the FileDescription record list, then it will also be used as a search field, in addition to the document's file name and directory. The additional identifiers can be defined inside the "Special Attributes" property group.

```
Dim FileDescription As SmRecList.SmRecordList
```

```
Dim RetCode As Integer

Dim ObjectId As Integer

Dim ClassId As Integer


FileDescription = SmRecList.NewRecordList


FileDescription.AddHeader "FILE_NAME",256,TDMT_CHAR
FileDescription.AddHeader "INTEGRATION_BEHAVIOR",256,TDMT_CHAR
FileDescription.AddHeader "TDM_COMPONENT_NAME",256,TDMT_CHAR
FileDescription.Value("FILE_NAME",0) = "c:\work\p01.sldprt"
FileDescription.Value("INTEGRATION_BEHAVIOR",0) = "TDM_SW_PART"
FileDescription.Value("TDM_COMPONENT_NAME",0) = "Default"


RetCode = SmCadInterface.FindObject(FileDescription, ObjectId, ClassId)
```

CADInterface Task: Opening a Document's Profile Card

The **Locate** method enables you to open the active document's profile card, as shown below:

```
Dim RetCode As Integer

RetCode = SmCADInterface.Locate(FileDescription)
```

CADInterface Task: Performing Life Cycle Operations

The following methods enable you to perform life cycle operations on a document:

- CheckIn
- CheckOut
- Release
- NewRelease
- Obsolete

The syntax for the life cycle operation methods is shown below:

```
Dim RetCode As Integer

RetCode = SmCADInterface.CheckIn(FileDescription)

RetCode = SmCADInterface.CheckOut(FileDescription)

RetCode = SmCADInterface.Release(FileDescription)

RetCode = SmCADInterface.NewRelease(FileDescription)

RetCode = SmCADInterface.Obsolete(FileDescription)
```

CADInterface Task: Managing SmarTeam API Windows

The **SetMainWindowHandle** method defines the main application window handle as the owner window of all SmarTeam API windows. Setting main window handle ensures that window management problems, such as those caused by the use of accelerator keys, are avoided, and that all SmarTeam windows are correctly closed on application termination.

The following example shows how to use this method:

```
Dim RetCode As Integer

Dim Wnd As HWND

Set Wnd = WinFind("Microsoft Word")

If Wnd Is Nothing Then

    MsgBox "Word Application is not active"

Else

    RetCode = SmCADInterface.SetMainWindowHandle(Wnd)

End If
```

CADInterface Task: Updating CAD File Properties in a Life-Cycle Operation

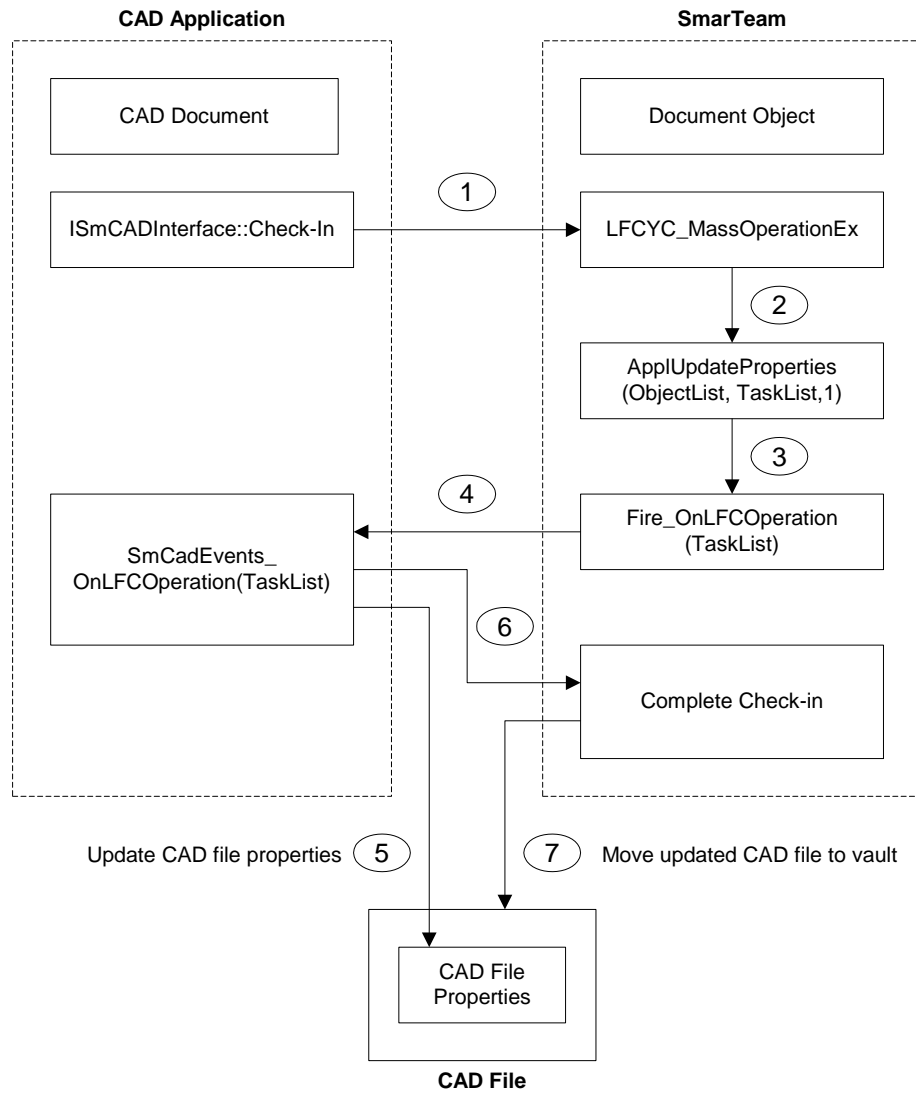
As mentioned above, the CAD document property information is stored in the CAD file's property fields. Among these file properties is revision and life-cycle state information.

When a SmarTeam life-cycle check in or release operation is performed on an object, the object's associated file is moved to the appropriate secured vault. During the life-cycle operation, the SmarTeam life-cycle mechanism can move a file to a vault but it does not have the ability to perform changes on the file's property fields, in particular it cannot update the file's revision and state information.

The SmCADInterface object provides a function ApplUpdateProperties and an event OnLFCOperation, which enables you to write a function in the CAD application that receives updated revision and state information from SmarTeam and updates the CAD file properties as a SmarTeam life-cycle operation is taking place.

Checking In an Object

For example, the following figure shows the sequence of operations that occurs when the CAD application checks in a CAD object.



The figure shows the following sequence of operations:

- The Check In API method is called in the CAD application, causing SmarTeam to begin checking in the corresponding SmarTeam object.
- The SmarTeam check in function calls the SmCADInterface method ApplUpdateProperties (ObjectList, TaskList,1) to transfer the TaskList. This method is called only if the SmCADInterface service is included in the session.
- ApplUpdateProperties causes SmCADInterface to fire an event - OnLFCOperation (TaskList).
- The event activates the CAD Application function SmCadEvents_OnLFCOperation (TaskList).

Using information from the TaskList, the CAD Application sets the life-cycle revision and state properties in the CAD file to coordinate with the properties in the corresponding SmarTeam object, before the CAD file is moved to the secured vault.

Control is returned to the SmarTeam check in function to complete the check in process.

The CAD file is moved to the secured vault.

Example

The following is an example of a function in the CAD application, which is activated by the OnLFCOperation event. It updates the CAD file properties.

```
Private Sub SmCadEvents_OnLFCOperation(ByVal TaskList As Object)

    Dim Cnt As Long

    Dim i As Long

    Dim SmTaskList As SmRecList.SmRecordList

    Dim FileDescription As SmRecList.SmRecordList

    Dim FileName As String

    Dim TempFileName As String

    Dim Directory As String

    Dim FullPath As String

    Dim oSmClass As ISmClass
```

```
Dim IsCheckedIn As Boolean

Dim CurrObjId As Long

Dim intClassID As Integer

Dim FileDesc As SmRecList.SmRecordList

Dim lngObjectID As Long

Dim TDMRet As Integer


ST_SetUpFileDescription FileDesc

ST_ActiveDocs_UpdateFileDesc FileDesc

TDMRet = SmCADInterface.FindObject(FileDesc, lngObjectID, intClassID)

Set SmTaskList = TaskList

Cnt = SmTaskList.RecordCount

For i = 0 To Cnt - 1

    FileName = SmTaskList.ValueAsString("CURR_FILE_NAME", i)

    If Len(FileName) > 0 Then

        Directory = SmTaskList.ValueAsString("CURR_DIRECTORY", i)

        FullPath = Directory & "\" & FileName

        CurrObjId = SmTaskList.ValueAsInteger("OBJECT_ID", i)

        If ObjectID = CurrObjId Then

            ' Set the revision attributes

            Set FileDescription = SmApplication.NewRecordList

            FileDescription.CopyRecord SmTaskList, i, 0

            ' Update CAD file properties

            SetProps_FileDesc_to_IA FileDescription

            ' Check if document is checked in

            IsCheckedIn = GetCheckedInDoc(SmTaskList, i)

            If IsCheckedIn = True Then

                ' Copy the document back to the vault
```

```
        TempFileName = "Temp" & ActiveWorkbook.Name

        ActiveWorkbook.SaveAs FileName:=TempFileName, AddToMRU:=False

        Directory = ActiveWorkbook.Path

        ST_FileClose "NoSave"

        ReplaceCheckedInDoc SmTaskList, i, Directory, TempFileName

    End If

    If IsCheckedIn = False Then

        ST_FileClose "Save"

    End If

End If

End If

Next i

End Sub
```

CADInterface Task: Invoking a User-Defined Command From the CAD Application

You can define a user-defined SmarTeam command by a script and invoke it from a CAD application using the function:

```
InvokeScriptWithList(SmRecList, Command)
```

The parameter SmRecList is loaded with the object attributes you want to be input to the script. This parameter is equivalent to the input parameter FirstParam in the normal usage of SmarTeam script hooks.

The parameter Command is the name of the script, as installed in SmarTeam by the Script Maintenance utility.

Calling this function from the CAD application is equivalent to activating a user-defined command from a SmarTeam menu. The difference is that a SmarTeam user command is associated with a specific SmarTeam object and the user command script automatically receives the object's attributes in its FirstParam input record list. When you use InvokeScriptWithList from a CAD application, there is no automatic association with a SmarTeam object so you need to create and fill the record list that will be used as FirstParam when the script is activated.

Example

The following example code is placed in the CAD application. It prepares the record list parameter RecList that will be used by the script as input parameter FirstParam. The code then calls InvokeScriptWithList, which runs the named script with the prepared record list as FirstParam.

```
Dim RetCode as Integer

Dim RecList as SmRecList.SmRecordList

Dim ScriptName as String

ScriptName = "MyScript"

' Prepare the input record list

Set RecList = new SmRecList.SmRecordList
```

```
' Set required headers and values inside the input RecList...  
  
.  
.  
.  
  
' Invoke the script  
  
RetCode = SmCADInterface.InvokeScriptWithList(RecList, ScriptName)
```

CADInterface Task: Improving Performance of Mass Save Operations

Two functions `BeginSaveOperation` and `EndSaveOperation` are provided to improve performance of the `CADInterface` for mass save operations using `Save` and `ODMA Save`.

The functions `BeginSaveOperation` and `EndSaveOperation` are used before and after a mass save operation, for example:

```
IntgToolLib.BeginSaveOperation  
  
    IntgToolLib.Save  
  
IntgToolLib.EndSaveOperation
```

Where the library function `IntgToolLib.Save` might perform a save of an entire assembly with all of its parts.

The effect of these operations is to minimize accesses to the `SmarTeam` data base and optimize operations in memory by retaining common information in memory during the save.

CADInterface Task: Improving Search Performance

The function

```
GetChildrenWithCopies(ClassId, ObjId, SelectAttributeList, ChildrenList)
```

retrieves all immediate children of an object with a single access to the database. The information about the children is stored in memory in `ChildrenList`. You can now search the `ChildrenList` for information efficiently without using multiple accesses to the database.

Example

The following is an example using the function `GetChildrenWithCopies`. The example loads all children of an object into the `ChildrenList` record list in memory and searches it for a child with the file name `Component.FileName`. Once found, other information about the child such as Object ID can be obtained.

' Calling function

```
Private Function SaveAssyStruct
```

```
    Dim ChildrenList As SmRecList.SmRecordList
```

```
    Set ChildrenList = New SmRecList.SmRecordList
```

```
    ' Load children into ChildrenList
```

```
    GetChildren(ChildrenList)
```

```
    Cnt = Children.Count
```

```
    ' Search for child according to file name
```

```
    For I=0 to Cnt
```

```
        Found = FindChild(Component(I).FileName, ChildrenList)
```

```
        If Not Found then
```

```
            'Look For the component inside the database
```

```
        End if
```

```
    End for
```

```
End Function
```

```
Private Function GetChildren(ChildrenList As SmRecList.SmRecordList) As  
Integer
```

```
    Dim RetCode As Integer
```

```
    Dim SelectList As SmRecList.SmRecordList
```

```
    Dim SmCadInterface As SmCadInterface
```

```
    Set SelectList = New SmRecList.SmRecordList
```

```
' Add attributes to the Additional attributes list

SelectList.AddHeader "COL_NAME", 256, sdtChar

SelectList.SetValueAsString "COL_NAME", 0, "FILE_NAME"

SelectList.SetValueAsString "COL_NAME", 1, "STATE"

' Call GetChildrenWithCopies

RetCode = SmCadInterface.GetChildrenWithCopies(AssyClassId, AssyObjId,
SelectList, ChildrenList)

GetChildren = RetCode

End Function

Private Function FindChild(FileName As String, ChildrenList As
SmRecList.SmRecordList) As Boolean

    Dim Found As Boolean

    Dim ChildFileName As String

    Dim State As Long

    Dim i As Integer

    Dim Cnt As Integer

    Found = False

    Cnt = ChildrenList.RecordCount

    For i = 0 To Cnt

        State = ChildrenList.ValueAsInteger("STATE", i)

        If State = 0 Or State = 2 Then ' new or checked out

            ChildFileName = ChildrenList.ValueAsString("FILE_NAME", i)

        Else

            ChildFileName = ChildrenList.ValueAsString("COPY_FILE_NAME", i)

        End If

    End For
```



```
        If LCase$(ChildFileName) = LCase$(FileName) Then
            Found = True
            Exit For
        End If
    Next

    FindChild = Found
End Function
```

10. SmIntegrationTool Library

Introduction

The SmIntegrationTool library enables you to perform the following functions:

- Define default class and File Type for the Integration Behaviors
- Set up mappings between CAD file property fields and SmarTeam class attributes in an integration

SmarTeam CAD Integration

The SmarTeam Integration Tool, by enabling you to map objects of an integrated CAD tool such as SolidWorks™ or Microsoft Word™ to objects in SmarTeam, lets you apply the power of SmarTeam to the CAD tool.

As shown in the following figure, the different CAD file types are mapped to SmarTeam classes and the CAD file property fields are mapped to the corresponding SmarTeam class attributes. For example, when a SolidWorks Part CAD file is mapped to the SmarTeam class SolidWorksPart, the Summary Information/Author property field of the CAD file can be mapped to the Object ID class attribute of the SolidWorksPart class. Now you can use that field in the CAD file for the SolidWorksPart User Object ID.

In addition, the CAD file is also linked to the corresponding SmarTeam class as an associated file; SmarTeam can manage the disposition of the CAD file in the SmarTeam vaults.

Chapter [9](#) discusses the SmarTeam CAD Interface library, which can be used once the SmarTeam integration is in place and the mappings established.

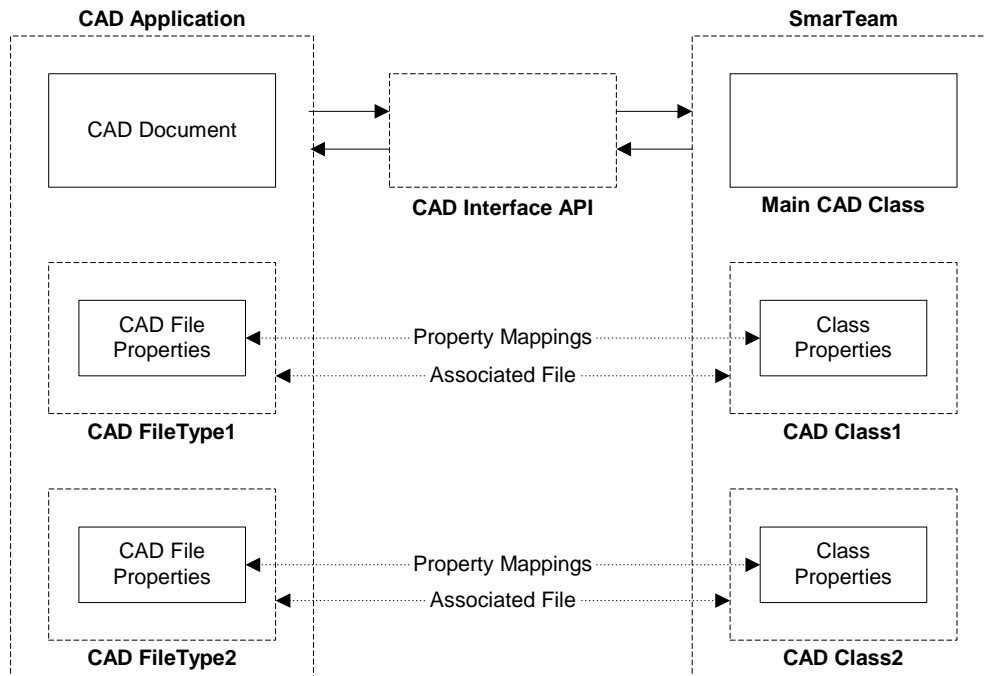


Figure 10-1 SmarTeam CAD Integration Overview of Objects

This section presents an overview of the main SmarTeam IntegrationTool objects including a description of the associated objects that are useful for the programmer:

- ISmIntegrationStore Object
- ISmCadFileTypes Object
- ISmPropertyGroupTypes Object
- ISmIntegrationGUIStore

ISmIntegrationStore

The ISmIntegrationStore object is the highest level object in the library. It contains objects for all SmarTeam Integrations. For each specific SmarTeam Integration, for example, the SolidWorks Integration, ISmIntegrationStore contains a ISmSpecificIntegrationStore object, which includes the objects for that Integration.

Object Diagram

The ISmIntegrationStore object and its major objects are shown in the following object diagram:

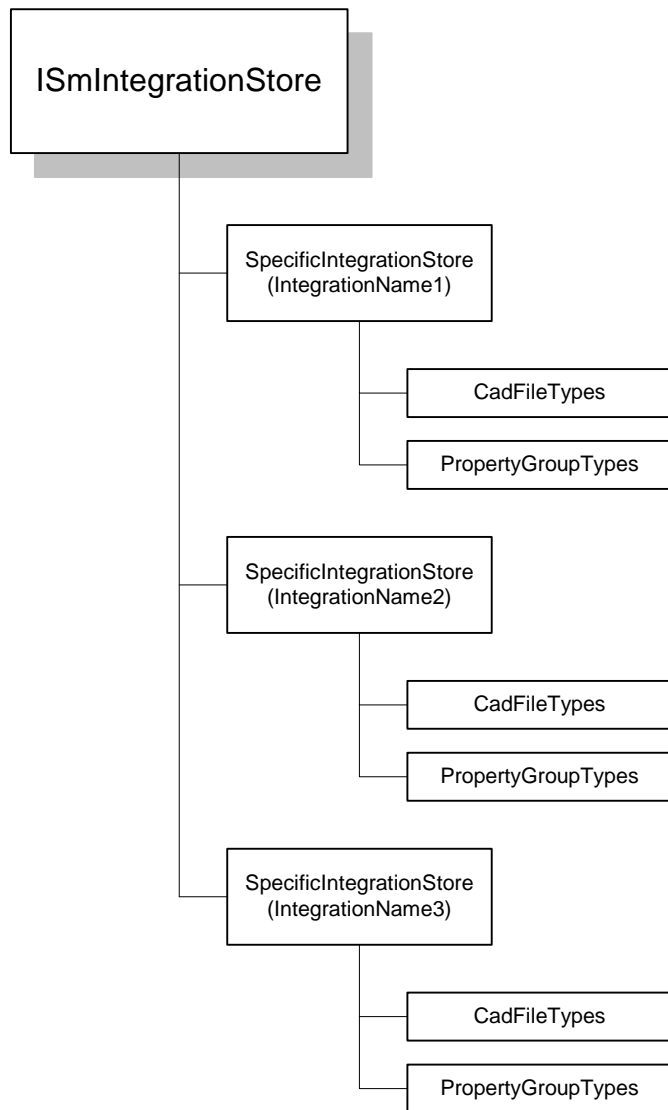


Figure 10-2 ISmIntegrationStore Object Diagram

Properties

The ISmIntegrationStore object contains the major properties:

Property	Description
Session	Returns an SmSession object that represents the parent session.
SpecificIntegrationStore	Returns the SmSpecificIntegrationStore object for Integratio

Methods

The ISmIntegrationStore object contains the methods:

Method	Description
NewPropertyGroup (PropertyGroupType)	Creates a new property group under the property group typ
NewGroupProperty (PropertyGroup)	Creates a new group property inside the PropertyGroup
GetGroupTypesForApplication (IntegrationName)	Returns all property group types that are defined for the application
NewGroupPropertyMapping (GroupProperty)	Creates a new attribute mapping for the GroupProperty
GetFileTypesForIntegration (IntegrationName)	Return all SmCadFileTypes that are defined for the applica
GetApplicationName (IntegrationName)	Returns the application (integration) name
Get_SpecificIntegrationStore (IntegrationName)	Return the SmSpecificIntegrationStore for the application. If modifications can be done inside the SmCadFileTypes after this call.
GetIntegrationGUIStore	Returns the SmIntegrationGUIStore, which is used to displa dialogs form the integration tools utility.
IntegrationRegistered (IntegrationName)	Returns true in case the integration is defined inside the database
GetIntegrationRegistry (IntegrationName)	Returns the SmCADIntegrationRegistry for the application

ISmSpecificIntegrationStore

Properties

The ISmSpecificIntegrationStore object contains the major properties:

Property	Description
Integration Name	The name of the specific SmarTeam Integration, for example, SolidWorks
CadFileTypes	Types of CAD files for this Integration
PropertyGroupTypes	Returns an SmPropertyGroupTypes collection object representing the integration mapping group types.
PropertyGroups	Returns an SmIntegrationPropertyGroups collection object representing the integration mapping groups.
GroupProperties	Returns an SmIntegrationGroupProperties collection object representing the integration mapping properties.
ClassesMappings	Returns an SmIntegrationClassesMappings collection object representing the integration mappings attributes.
CadFileTypes	Returns an SmCadFileTypes collection object representing the integration supported file types.
ManagedClasses	Returns an SmIntegrationManagedClasses collection object representing the integration managed classes.
DefaultManagedClasses	Returns an SmIntegrationManagedClasses collection object representing the integration default managed classes.
IntegrationStore	Returns the ISmIntegrationStore object

Methods

The ISmSpecificIntegrationStore object contains the methods:

Method	Description
SmClassMappingByAttribute (SmClassAttribute, GroupProperty)	Returns a ISmClassMapping for SmClassAttribute and GroupProperty
AllMappingsForPropertyName (PropertyName)	Returns a ISmIntegrationClassesMappings object for Property
SmCadFileType (FileTypeId)	Returns the SmCadFileType according to the file type object id
AllMappingsForClassByGroupName (GroupName, ClassId)	Returns the ISmIntegrationClassMappings object for Group and ClassId
GetManagedClassesForBehavior (Integration Behavior)	Retrieves classes that support the specified Integration Behavior
GetDefaultManagedClassForBehavior (Integration Behavior)	Retrieves the default class assigned for the specified Integration Behavior
ManagedClassesForCadFileType (SmCadFileType)	Returns ISmManagedClasses for CadFileType.
DefaultManagedClassForCadFileType (SmCadFileType)	Returns the ISmManagedClass for SmCadFileType
MappingsForGroupProperty (SmGroupProperty)	Returns the ISmClassesMappings for Group Property
PropertiesForGroup (SmPropertyGroup)	Returns the ISmGroupProperties for PropertyGroup
GroupsForGroupType (SmPropertyGroupType)	Returns the ISmPropertyGroups for PropertyGroupType

Correspondence with Integration Tool

The above objects are shown on the Integration Tool screen in Figure 10-3. The correspondence between objects on the screen and objects in the SmIntegrationTool Library are described in the table following the figure.

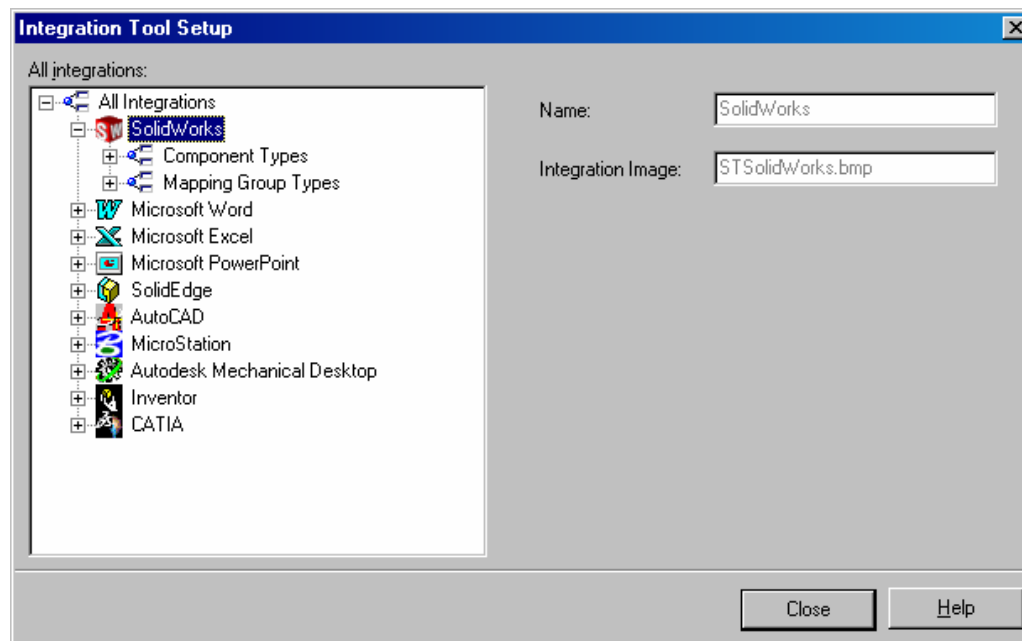


Figure 10-3 Integration Tool Screen

Object on Integration Tool Screen	Object in SmIntegrationTool Library
All Integrations	ISmIntegrationStore
SolidWorks	SpecificIntegrationStore
Supported component types	CadFileTypes
Mapping group types	PropertyGroupTypes
AutoCad	SpecificIntegrationStore
Supported component types	CadFileTypes
Mapping group types	PropertyGroupTypes
...	

ISmCadFileTypes

A ISmCadFileTypes object is a collection of ISmCadFileType objects and represents all mappings of Integration file types to the corresponding SmarTeam managed classes.

Object Diagram

The ISmCadFileTypes components and their corresponding objects are shown in the following object diagram:

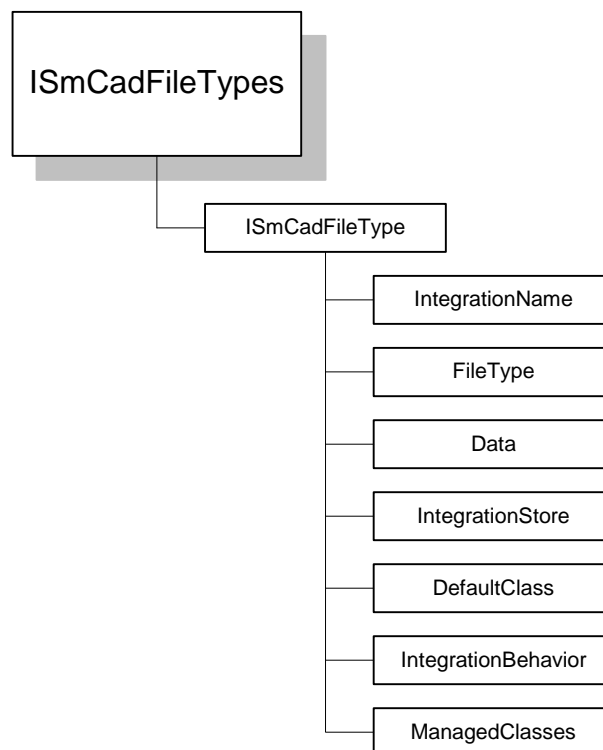


Figure 10-4 ISmCadFileTypes Object Diagram

Obtaining the ISmCadFileTypes Object

To obtain an ISmCadFileTypes Object:

```
CadFileTypes =  
IntegrationStore.SpecificIntegrationStore(IntegrationName1).CadFileTypes
```

Methods

The ISmCadFileTypes object has the following methods

Method	Description
GetItemByBehavior	Retrieves the Default File Type for the specified Integration Behavior

ISmCadFileType

The ISmCadFileType object represents CAD specific component types.

Properties

The ISmCadFileType object has the following properties

Property	Description
IntegrationName	Returns the associated integration name.
FileType	Returns an SmLookupObject object representing the referenced file type
Data	Returns an SmRecord object that represents object's data.
ManagedClasses	Returns an SmManagedClasses collection object representing object's managed classes
IntegrationStore	Returns an SmIntegrationStore object representing the associated integration store.
DefaultClass	Returns an SmManagedClass object representing object's default class.
IntegrationBehavior	Represents an integration behavior specific for the component type. See CADInterface for examples of use.

ISmManagedClasses

The ISmManagedClasses object represents the set of SmarTeam managed classes to which a specific integration behavior is mapped.

Object Diagram

The ISmManagedClasses components and their corresponding objects are shown in the following object diagram:

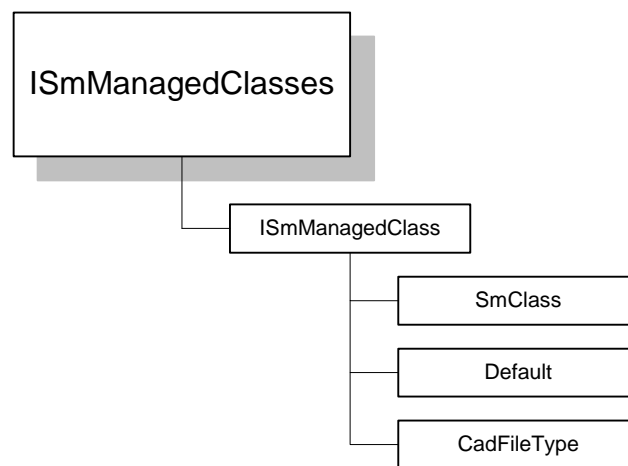


Figure 10-5 ISmCadFileTypes Object Diagram

ISmManagedClass

The ISmManagedClass object represents an individual SmarTeam class to which the Integration Behavior is mapped.

Properties

ISmManagedClass object has the properties:

Property	Description
SmClass	Returns the ISmClass object which is linked to the managed class
Default	True if object represents a default managed class (the class that is first displayed inside the profile card, when adding a new object).
CadFileType	Returns an SmCadFileType object that represents the parent supported type.

Methods

The ISmManagedClass object has the following methods

Method	Description
Save	Saves managed class to the database.

Correspondence with Integration Tool

The above objects are shown on the Integration Tool screen in Figure 10-6. The correspondence between objects on the screen and objects in the SmIntegrationTool Library are described in the table following the figure.

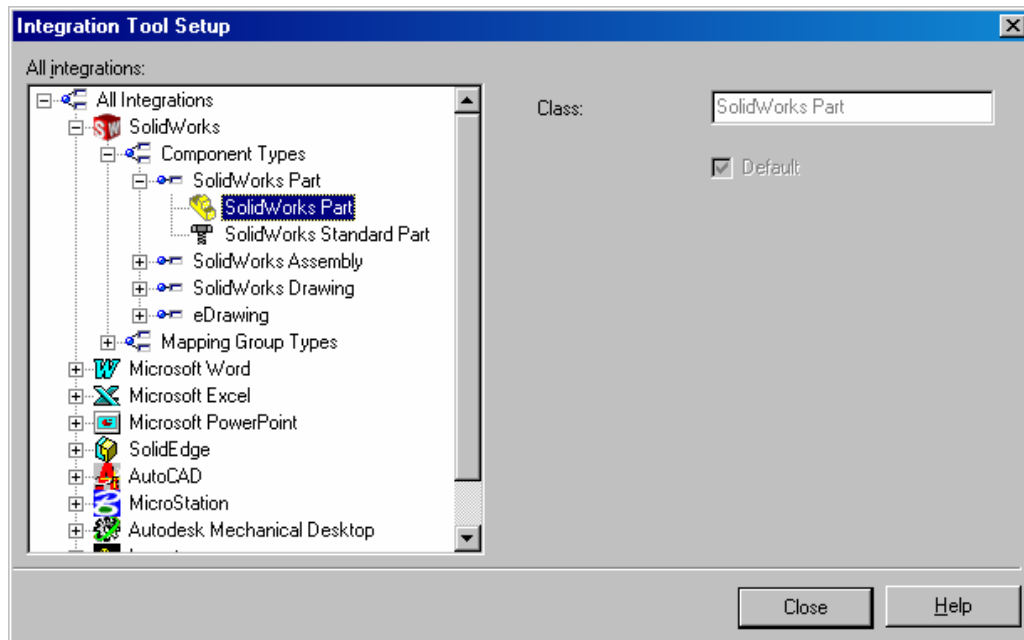


Figure 10-6 ISmCadFileType

Object on Integration Tool Screen	Object in SmIntegrationTool Library
All Integrations	ISmIntegrationStore
SolidWorks	ISmSpecificIntegrationStore
Supported component types	ISmCadFileType
SolidWorks Part	ISmCadFileType
Classes mapped to SolidWorks Part:	ISmManagedClasses
SolidWorks Part	ISmManagedClass
Document	ISmManagedClass
SolidWorks Assembly	ISmCadFileType
Classes mapped to SolidWorks Assen	ISmManagedClasses
SolidWorks Assembly	ISmManagedClass
SolidWorks Drawing	ISmCadFileType
Classes mapped to SolidWorks Drawin	ISmManagedClasses
SolidWorks Drawing	ISmManagedClass
eDrawing	ISmCadFileType
Classes mapped to eDrawing:	ISmManagedClasses
eDrawing	ISmManagedClass
. . .	

ISmPropertyGroupTypes

A ISmPropertyGroupTypes object is a collection of ISmPropertyGroupType objects and represents all mappings of all types of Integration file property fields to the corresponding SmarTeam managed class attributes.

Object Diagram

The ISmPropertyGroupTypes components and their corresponding objects are shown in the following object diagram:

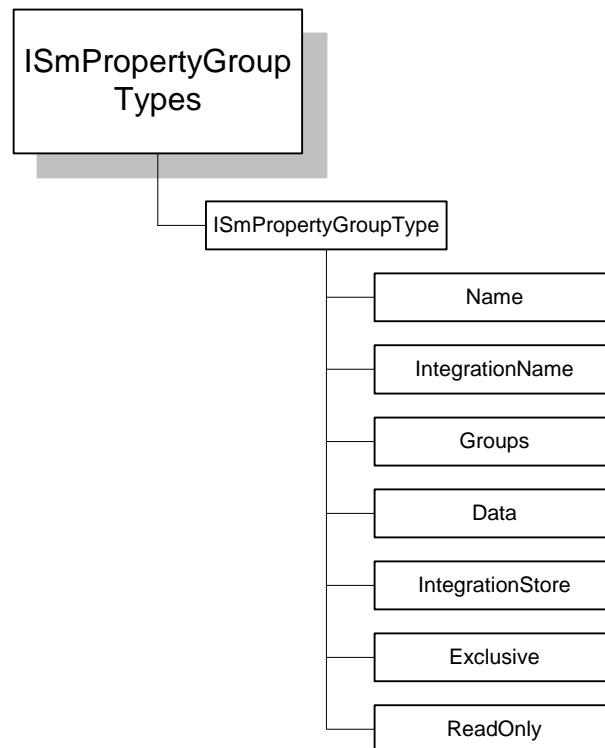


Figure 10-7 ISmPropertyGroupTypes Object Diagram

Obtaining the ISmPropertyGroupTypes Object

To obtain an ISmPropertyGroupTypes Object:

```
PropertyGroupTypes =  
IntegrationStore.SpecificIntegrationStore(IntegrationName1).PropertyGroupTypes
```

Adding a New ISmPropertyGroupType to the Collection

You use the AddPropertyGroupType (const GroupTypeName: WideString, Exclusive: WordBool, ReadOnly: WordBool): ISmRegisteredPropertyGroupType method of the ISmCADIntegrationRegistry object to add a new ISmPropertyGroupType object to the ISmPropertyGroupTypes collection.

AddPropertyGroupType Method

The AddPropertyGroupType method is called as follows:

```
Set IntegrationRegistry = ISmIntegrationStore.  
CreateIntegrationRegistry(IntegrationName1, Image)  
  
RegisteredPropertyGroupType =  
IntegrationRegistry.AddPropertyGroupType(GroupTypeName, Exclusive, ReadOnly)
```

The arguments of the method are:

Argument	Description
GroupTypeName	Name for the property group (hard coded)
Exclusive	True if group type is exclusive.
ReadOnly	True if group type is read only.

See the example in the section ISmCadFileTypes.

ISmPropertyGroupType

The ISmPropertyGroupType object represents all mappings of an specific type of Integration file property field to a set of SmarTeam classes.

Properties

The ISmPropertyGroupType object has the following properties

Property	Description
Name	Returns the group type name.
IntegrationName	Returns the associated integration name.
Groups	Returns an SmPropertyGroups collection object representing the object's mapping groups.
Data	Returns a SmRecord object that represents object's data.
SmIntegrationStore	Returns a SmIntegrationStore object representing the associated integration store.
Exclusive	True if group type is exclusive, i.e., no more than one group exists inside group type.
ReadOnly	True if group type is read only.

Correspondence with Integration Tool

The above objects are shown on the Integration Tool screen in Figure 10-8. The correspondence between objects on the screen and objects in the SmIntegrationTool Library are described in the table following the figure.

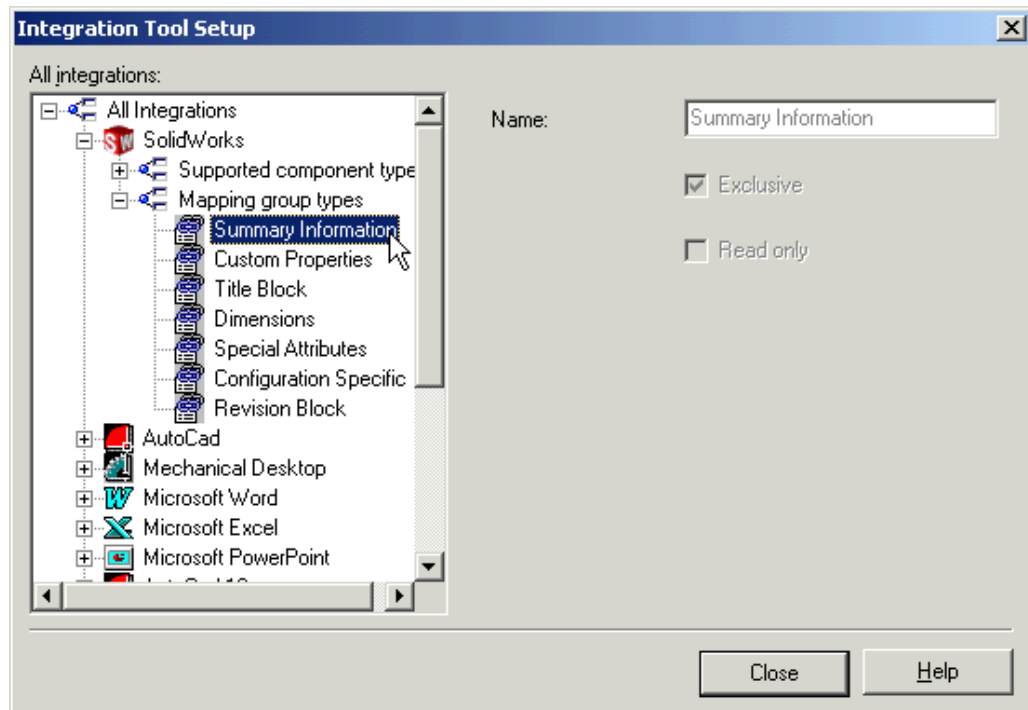


Figure 10-8 Mapping Group Types

Object on Integration Tool Screen	Object in SmIntegrationTool Library
All Integrations	ISmIntegrationStore
SolidWorks	SpecificIntegrationStore
Supported component types	CadFileTypes
Mapping group types	PropertyGroupTypes
Summary Information	PropertyGroupType
Custom Properties	PropertyGroupType
Title Block	PropertyGroupType
. . .	

ISmPropertyGroups

An ISmPropertyGroups object is a collection of ISmPropertyGroup objects and represents all groups of mappings of a specific type of Integration file property field to the corresponding SmarTeam managed class attributes.

Object Diagram

The ISmPropertyGroupTypes components and their corresponding objects are shown in the following object diagram:

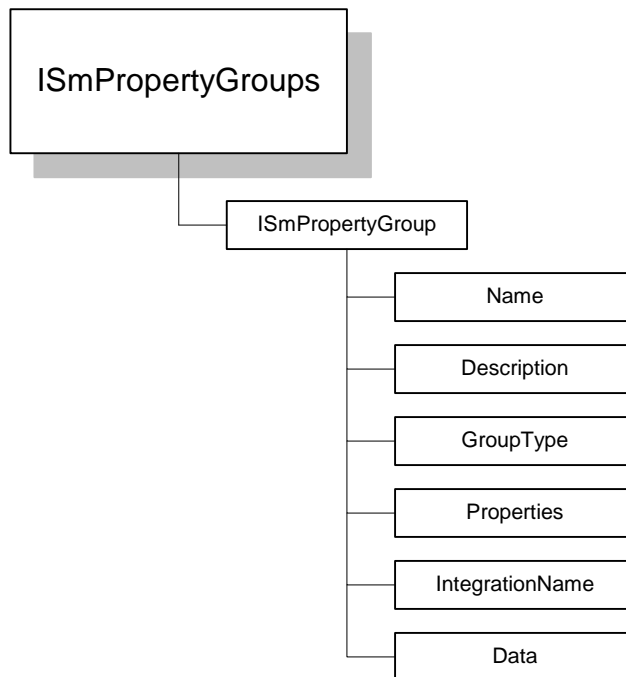


Figure 10-9 *ISmPropertyGroups* Object Diagram

Obtaining the `ISmPropertyGroups` Object

To obtain an `ISmPropertyGroups` Object:

```
PropertyGroups =  
IntegrationStore.SpecificIntegrationStore(IntegrationName1).PropertyGroupType.  
Groups
```

Adding a New `ISmPropertyGroup` to the Collection

You use the `AddPropertyGroup(const GroupTypeName: WideString, Exclusive: WordBool, ReadOnly: WordBool): ISmRegisteredPropertyGroup` method of the `ISmCADIntegrationRegistry` object to add a new `ISmPropertyGroup` object to the `ISmPropertyGroups` collection.

AddGroup Method

The AddGroup method is called as follows:

```
Set IntegrationRegistry = ISmIntegrationStore.  
CreateIntegrationRegistry(IntegrationName1, Image)  
  
RegisteredPropertyGroupType =  
IntegrationRegistry.AddPropertyGroupType(GroupTypeName, Exclusive, ReadOnly)  
  
Set ISmRegisteredPropertyGroup =  
RegisteredPropertyGroupType.AddGroup(GroupName, GroupDescription)
```

The arguments of the method are:

Argument	Description
GroupName	Name for the property group (hard coded)
GroupDescription	Description for the property group.

Add Method

Alternatively you can use the Add method of ISmPropertyGroups as follows:

```
Get PropertyGroupType  
  
Set SmPropertyGroup = ISmIntegrationStore.  
NewPropertyGroup(PropertyGroupType):  
  
ISmPropertyGroups.Add(SmPropertyGroup)
```

Example

See the example in the section ISmCadFileTypes.

ISmPropertyGroup

The ISmPropertyGroup object represents a mapping of a specific type of Integration file property field to a set of SmarTeam classes.

Properties

The ISmPropertyGroup object has the following properties

Property	Description
Name	Returns the group type name.
Description	Returns or sets the group description.
GroupType	Returns an SmPropertyGroupType object representing the parent mapping group type.
Properties	Returns an SmGroupProperties collection object representing the object's mapping properties.
IntegrationName	Returns the associated integration name.
Data	Returns an SmRecord object that represents object's data.

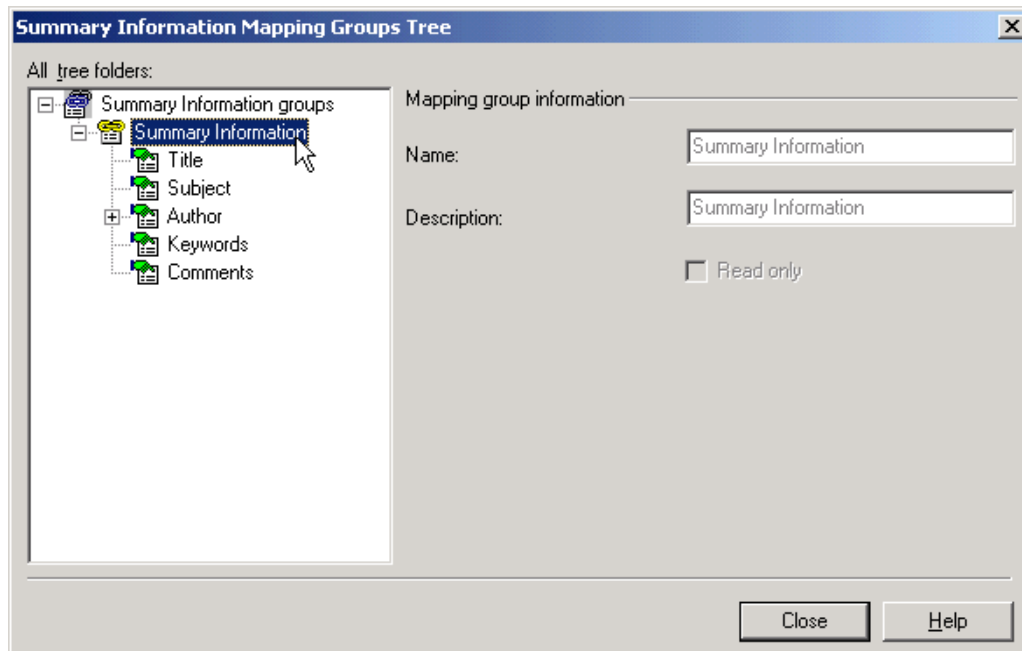
Methods

The ISmPropertyGroup object has the following methods

Method	Description
Save	Saves the property group to the database.

Correspondence with Integration Tool

The above objects are shown on the Integration Tool screen in Figure 10-10. The correspondence between objects on the screen and objects in the SmIntegrationTool Library are described in the table following the figure.

*Figure 10-10 Mapping a Property Group*

Object on Integration Tool Screen	Object in SmIntegrationTool Library
Summary Information groups	ISmPropertyGroups
Summary Information	ISmPropertyGroup
GroupProperties for this PropertyGroup:	ISmGroupProperties
Title	ISmGroupProperty
Subject	ISmGroupProperty
Author	ISmGroupProperty
Keywords	ISmGroupProperty
Comments	ISmGroupProperty

ISmGroupProperties

A ISmGroupProperties object is a collection of ISmGroupProperty objects and represents all properties of mappings of a specific group of Integration file property fields to the corresponding SmarTeam managed class attributes.

Object Diagram

The ISmGroupPropertyTypes components and their corresponding objects are shown in the following object diagram:

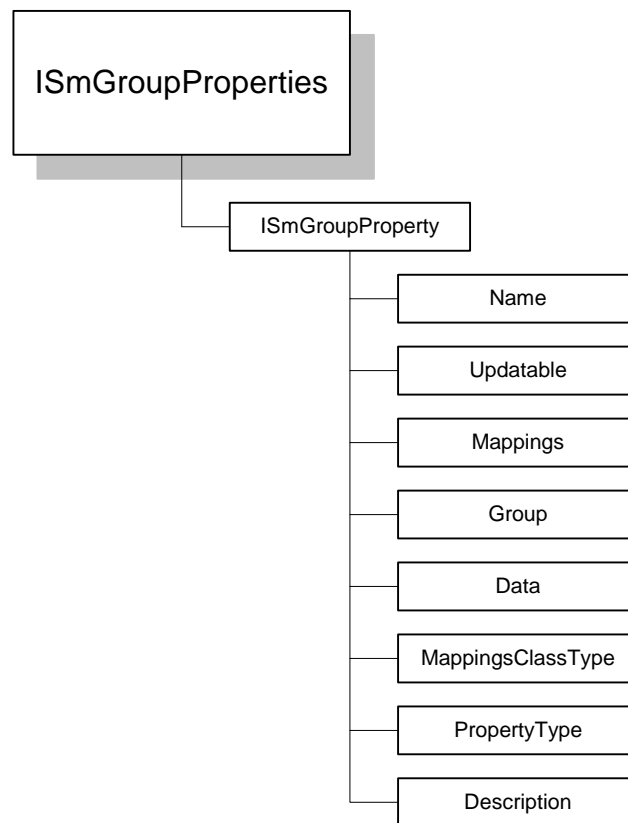


Figure 10-11 ISmGroupProperties Object Diagram

Obtaining the ISmGroupProperties Object

To obtain an ISmGroupProperties Object:

```
GroupProperties =  
IntegrationStore.SpecificIntegrationStore(IntegrationName1).GroupPropertyType.  
Groups(i).Properties
```

Adding a New ISmGroupProperty to the Collection

You use the Add method of the ISmGroupProperties object to add a new GroupProperty to the group.

Add Method

You can use the Add method of *ISmGroupProperties* as follows:

```
Get PropertyGroup  
  
Set SmGroupProperty = ISmIntegrationStore. NewGroupProperty(PropertyGroup):  
  
ISmGroupProperties. Add(SmGroupProperty))
```

Example

The following example shows how to add a property group to the *Mass Properties* group type.

```
Dim SmRegisteredPropertyGroupType as ISmRegisteredPropertyGroupType  
  
Dim SmRegisteredPropertyGroup as ISmRegisteredPropertyGroup  
  
Set SmRegisteredPropertyGroupType = SmCADIntReg.AddPropertyGroupType("Mass  
Properties", True, True)  
  
if Not SmRegisteredPropertyGroupType is Nothing then  
  
    Set SmRegisteredPropertyGroup =  
    SmRegisteredPropertyGroupType.AddGroup("Mass Properties", "Mass Properties")  
  
    If Not SmRegisteredPropertyGroup is Nothing then  
  
        SmRegisteredPropertyGroup.AddProperty    "Volume", "Volume"  
        TDMT_DOUBLE, ctAll, True  
  
        SmRegisteredPropertyGroup.AddProperty    "Area", "Area", TDMT_DOUBLE,  
        ctAll, True  
  
    End If  
  
End If
```

ISmGroupProperty

The ISmGroupProperty object represents a mapping of a specific type of Integration file property field to a set of SmarTeam classes.

Properties

The ISmGroupProperty object has the following properties

Property	Description
Name	Returns or sets property name.
Updatable	True if property can be updated by a SmarTeam attribute.
Mappings	Returns an SmClassesMappings collection object representing the object's mapping attributes.
Group	Returns an SmPropertyGroup object representing the parent mapp group.
Data	Returns an SmRecord object representing object's data.
MappingsClassType	Returns or sets possible mapped classes type.
PropertyType	Returns or sets the property type.
Description	Returns or sets the property description.

Methods

The ISmGroupProperty object has the following methods

Method	Description
Save	Saves group property to the database.

Correspondence with Integration Tool

The above objects are shown on the Integration Tool screen in Figure 10-10. The correspondence between objects on the screen and objects in the SmIntegrationTool Library are described in the table following the figure.

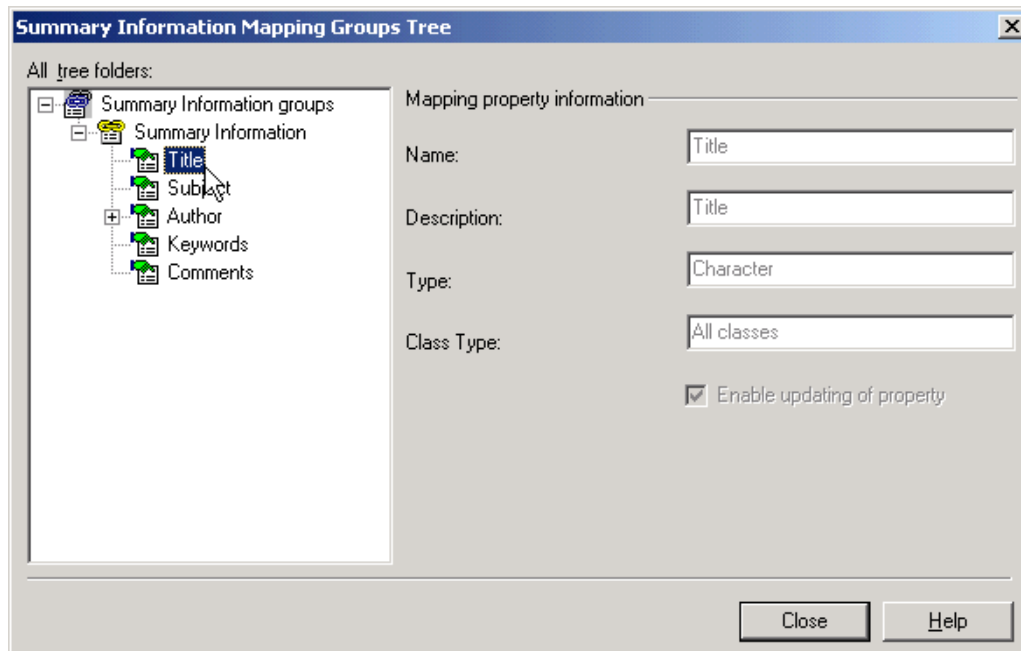


Figure 10-12 Mapping Group Properties

Object on Integration Tool Screen	Object in SmIntegrationTool Library
Summary Information groups	ISmPropertyGroups
Summary Information	ISmPropertyGroup
GroupProperties for this PropertyGroup	ISmGroupProperties
Title	ISmGroupProperty
Subject	ISmGroupProperty
Author	ISmGroupProperty
Keywords	ISmGroupProperty
Comments	ISmGroupProperty

ISmClassesMappings

A ISmClassesMappings object is a collection of ISmClassMapping objects and represents all mappings of a specific Integration file property field to the corresponding SmarTeam managed class attribute. This ISmClassesMappings object collects the mappings from this CAD file property field (GroupProperty) to a SmarTeam class attribute for all CAD files in this integration. For example, each of the SolidWorks CAD file types has an Author field. For each file type, the Author field is mapped to the User Object ID class attribute for the corresponding class, i.e. the Author field of the SW Assembly file is mapped to the User Object ID attribute of the SW Assembly class.

Object Diagram

The ISmClassMappingTypes components and their corresponding objects are shown in the following object diagram:

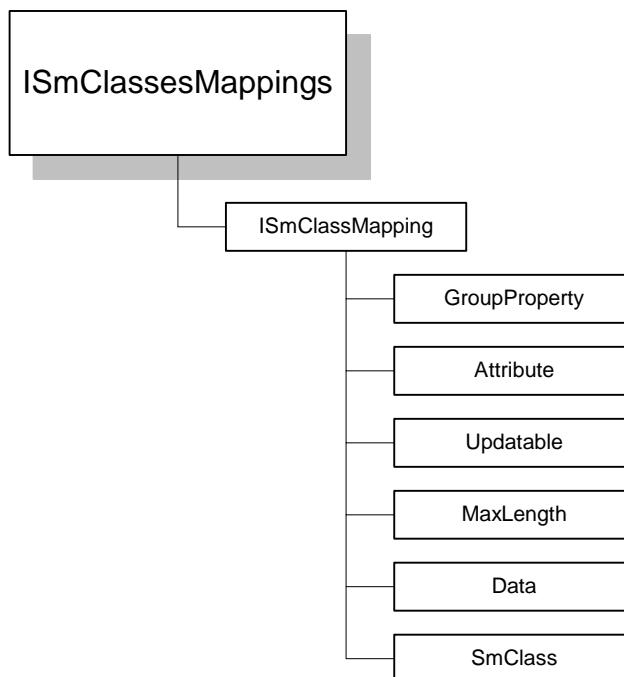


Figure 10-13 ISmClassesMappings Object Diagram

Obtaining the ISmClassesMappings Object

To obtain an ISmClassesMappings Object:

```
Mappings = IntegrationStore.SpecificIntegrationStore(IntegrationName1).  
GroupPropertyType.Groups(i).Properties(i).Mappings
```

ISmClassMapping

The ISmClassMapping object represents a mapping of an specific Integration file property field to a set of SmarTeam classes.

Adding a New ISmClassMapping

You can add a new ISmClassMapping as follows:

```
Get GroupProperty  
  
Set SmClassMapping = ISmIntegrationStore.  
NewGroupPropertyMapping(GroupProperty)  
  
SmClassMapping.SmClass = SmClass  
  
SmClassMapping.Attribute = SmClassAttribute  
  
SmClassMapping.MaxLength = SizeInt  
  
SmClassMapping.Updatable = True  
  
SmClassMapping.Save
```

Properties

The ISmClassMapping object has the following properties

Property	Description
GroupProperty	Returns an SmGroupProperty object representing the parent mapping property.
Attribute	Returns or sets the mapping attribute.
Updatable	True if SmarTeam attribute can be updated by the parent mapping property.
MaxLength	Returns or sets the maximum mapping size.
Data	Returns an SmRecord object representing object's data.
SmClass	Returns or sets an SmClass object representing the mapping class.

Methods

The ISmClassMapping object has the following methods

Method	Description
Save	Saves the class mapping to the database.

Correspondence with Integration Tool

The above objects are shown on the Integration Tool screen in Figure 10-14. The correspondence between objects on the screen and objects in the SmIntegrationTool Library are described in the table following the figure.

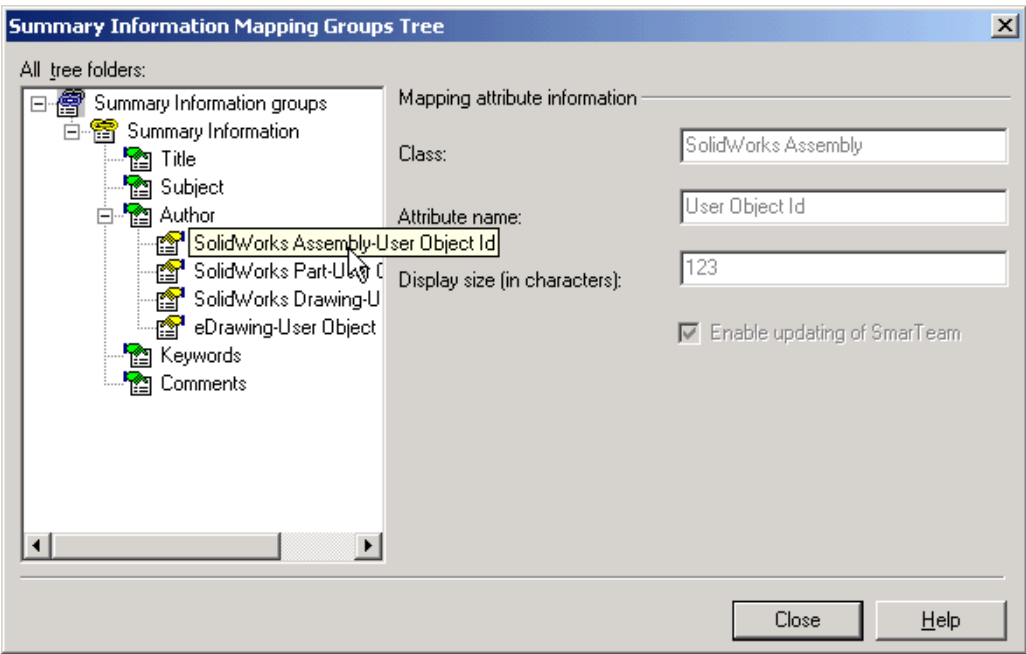


Figure 10-14 Mappings for Group Property

Object on Integration Tool Screen	Object in SmlIntegrationTool Library
Summary Information groups	ISmPropertyGroups
Summary Information	ISmPropertyGroup
GroupProperties for this PropertyGroup:	ISmGroupProperties
Title	ISmGroupProperty
Subject	ISmGroupProperty
Author	ISmGroupProperty
Class Mappings for this Group Property:	ISmClassesMappings
SolidWorks Assembly – User Object Id	ISmClassMapping
SolidWorks Part – User Object Id	ISmClassMapping
SolidWorks Drawing – User Object Id	ISmClassMapping
eDrawing – User Object Id	ISmClassMapping

ISmIntegrationGUIStore

The ISmIntegrationGUIStore object is used to display the objects in the application.

Object Diagram

The ISmIntegrationGUIStore object and its major objects are shown in the following object diagram:

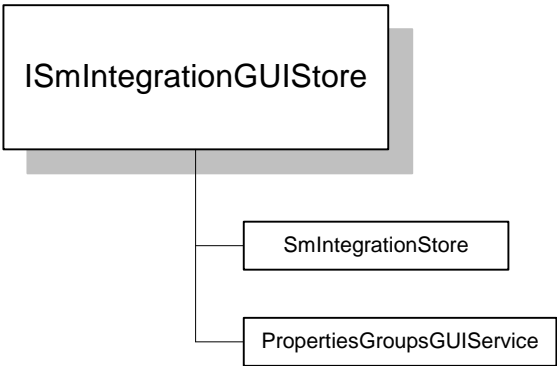


Figure 10-15 ISmIntegrationGUIStore Object Diagram

Properties

The ISmIntegrationGUIStore object contains the properties:

Property	Description
SmIntegrationStore	Returns the ISmIntegrationStore object
PropertiesGroupsGUIService	Returns ISmSpecificIntegrationStore.

Methods

The ISmIntegrationGUIStore object contains the methods:

Method	Description
Init(SmIntegrationStore)	Should be called right after the object was created
ObjectProfile(PossibleClasses, Attributes, AddToDesktop, ModalResult:)	Opens a profile card for new object
OpenClassManagementScreen (IntegrationName)	Opens the managed classes dialog

ISmPropertiesGroupsGUIService

Properties

The ISmPropertiesGroupsGUIService object contains the major properties:

Property	Description
SmIntegrationGUIStore	Returns the SmIntegrationGUIStore object

Methods

The ISmPropertiesGroupsGUIService object contains the methods:

Method	Description
OpenGroupAttributeView (SmPropertyGroup, AddMode)	Opens a view with the attribute mapping
OpenGroupPropertiesTree (SmPropertyGroup, MaxDepth)	Opens a tree view for a property group
OpenGroupPropertiesView (SmPropertyGroup, WasChanged)	Opens a view with the properties
OpenGroupPropertyAttributeView (SmGroupProperty, AddMode): ViewModalResultEnum	Opens a modal attribute view for a group property.
OpenGroupsTree (SmPropertyGroupType, MaxDepth)	Opens a tree view for the property group type
OpenGroupsView (SmIntegrationName, WasChanged)	Open groups view for a specific integration
OpenGroupsViewForSpecificGroupType (SmGroupType, WasChanged)	Opens a view with all groups under a group
OpenGroupTypesTree (IntegrationName, MaxDepth)	Opens a tree view with all group types for a specific integration
OpenMappingsTree	Opens a tree view with all mappings for a group

(SmGroupProperty, MaxDepth)	group property
OpenPropertyMappingAttributeView (SmMapping, AddMode, CanModifyClass)	Opens a modal view for the class mapping
OpenPropertyMappingsView (SmGroupProperty, WasChanged)	Opens a view with the mapping for a group property
OpenSpecificClassPropertyMappingsView (SmGroupProperty, ClassId, WasChanged)	Opens a view with mapping for a specific class

Example

This example shows how to display the integration tool objects.

```
SmPropertiesGroupsGUIService = SmIntegrationGUIStore.  
PropertiesGroupsGUIService  
  
` Display the Add Property dialog  
  
SmPropertiesGroupsGUIService.OpenGroupPropertyAttributeView(SmGroupProperty,  
True)  
  
` Display the Group Types tree for the application  
  
GUIServices.OpenGroupTypesTree("Microsoft Word", dpthPropertyMappings)
```

PART III

SMARTTEAM

CLIENT LIBRARIES

11. SmartIXF Library

Introduction

The **IXF** library enables you to perform the following functions:

- Generating an iXF schema
- Processing an iXF schema
- Generating an iXF Archive File
- Processing an iXF Archive File

The iXF format created and processed by the iXF library conforms to the format described in the “iXF Specifications 1.0” document, available at <http://www.ixfstd.org/std/docs/ixf>.

For additional information on the iXF format, see the iXF Standard web site, at <http://www.ixfstd.org>.

Naming Conventions

This section describes the naming conventions used in this guide.

NCName

A valid NCName must begin with a letter or an underscore (_) and cannot contain spaces; letters, digits, and underscores are allowed after the first character:

```
NCName ::= (Letter | '_') (NCNameChar)
```

```
NCNameChar ::= Letter | Digit | '.' | '-' | '_'
```

For example: “PartMaster”. See also <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

Class Behavior URI

A valid class behavior URI must contain a namespace and a behavior name, separated by “#” as: Class Behavior URI : : = (Namespace) (‘#’) (Name).

For example,

“<http://www.ixfstd.org/std/ns/core/classBehaviors/links/1.0#link>”

Overview of Objects

This section presents an overview of the main SmartIxf objects including a description of the associated objects that are useful for the programmer:

- ISmIxfSchema Object
- SmIxfWriter Object
- SmIxfReader Object
- ISmIxfStdHelper Object

ISmIxfSchema

The ISmIxfSchema object serves to organize and refer to the components of the iXF schema document.

The Schema is maintained wholly in memory. The Schema can be saved to a file in two ways:

Through SmIxfExternalSchemaWriter

When creating an archive file using the SmIxfWriter.

The ISmIxfSchema object is not a top-level object in the object hierarchy but is contained in the following top-level objects:

- [SmIxfWriter](#)– for writing the iXF schema document
-

-
- SmIxfReader – for reading the iXF schema document
 - [SmIxfExternalSchemaWriter](#) – for writing an external iXF schema document.
 - [SmIxfExternalSchemaReader](#)– for reading an external iXF schema document

Because of its importance, the ISmIxfSchema object is discussed separately in this major section; refer to the sections for the above objects for information on how the ISmIxfSchema object is included in those objects.

The schema components and their corresponding objects are shown in the following object diagram:

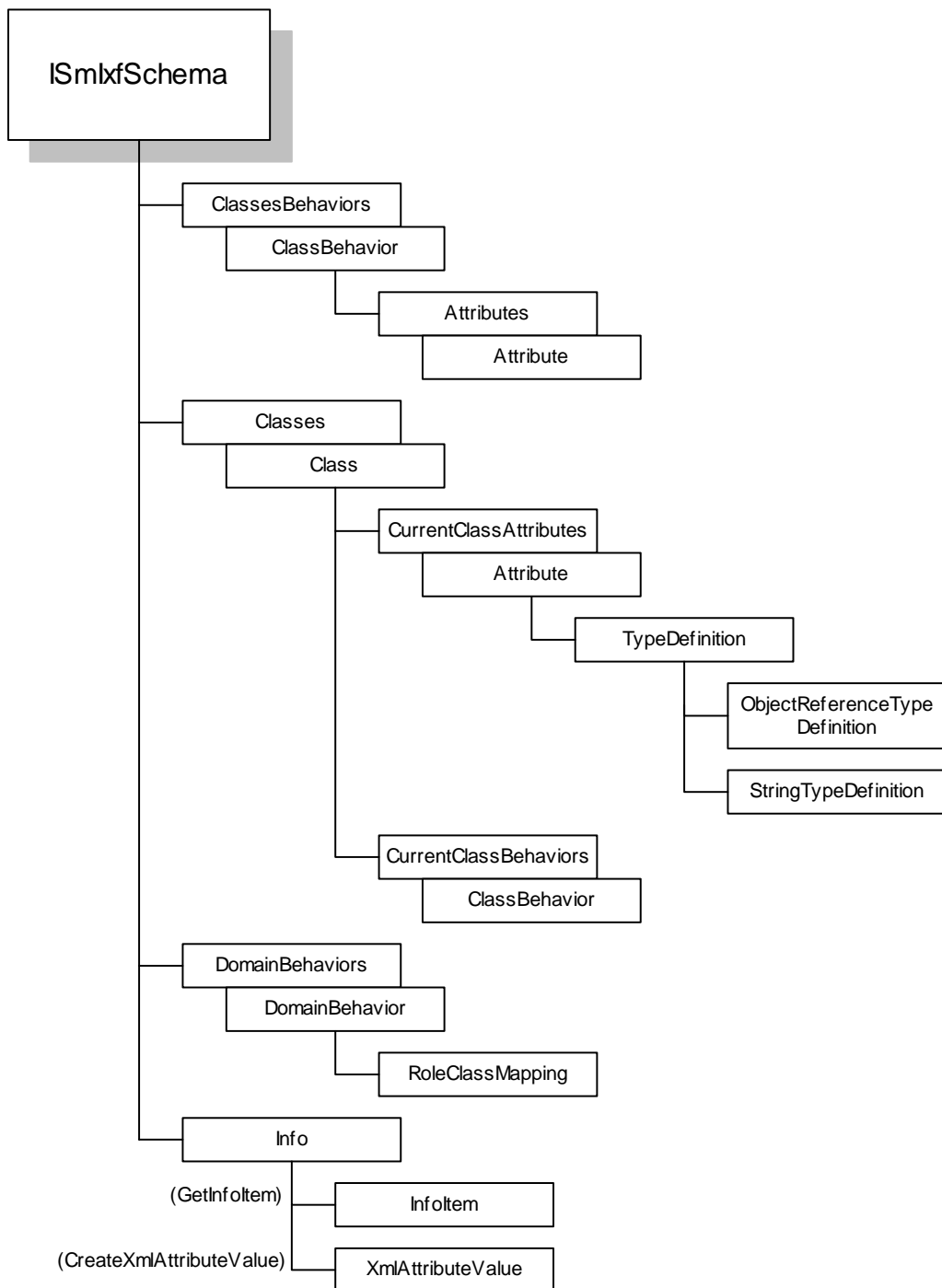


Figure 11-1 ISmxfSchema Object Diagram

Properties

The ISmIxfSchema object has the following properties

Property	Description
Classes	Collection ISmIxfClasses of schema classes.
ClassesBehaviors	Collection ISmIxfClassesBehaviors of schema ClassBehaviors
DomainBehaviors	Collection ISmIxfDomainBehaviors of schema DomainBehaviors
Info	ISmIxfInfo object for holding miscellaneous information
SchemaLocation	The physical location of the schema file
SchemaURI	The Schema URI, which is the unique identifier of the schema.

Obtaining the ISmIxfSchema Object

To create an ISmIxfSchema Object from the SmIxfWriter Object (to create a SmIxfWriter object see SmIxfWriter):

```
IxfWriter.Schema
```

A Schema object can also be obtained similarly from the SmIxfReader Object, SmIxfExternalSchemaWriter Object, and from the SmIxfExternalSchemaReader Object.

ISmIxfClassesBehaviors

An ISmIxfClassesBehaviors object is a collection of ISmIxfClassBehavior objects and represents all ISmIxfClassBehavior objects related to the IXF schema.

Note: This object is not the same as the ISmIxfClassBehaviors, which represents all ISmIxfClassBehavior objects declared by a specific class.

Adding a New ClassBehavior to the IXF Schema

You use the Add method of the ClassesBehaviors object to add a new ClassBehavior object to the IXF Schema. Once you have added a ClassBehavior to the ClassesBehaviors object you can declare it in a specific class. Before using the Add method, you need to understand how a ClassBehavior is packaged.

ClassBehavior Schema File

A ClassBehavior object is defined in a ClassBehavior schema file. The ClassBehavior schema file can be packaged either embedded in or external to the IXF Archive file as described below.

ClassBehavior Schema File Packaging State

The following table describes the packaging states of the ClassBehavior schema file and the software constant used for each state. The packaging state of the ClassBehavior schema file is described by the ModeTypeEnum parameter of the Add function.

Packaging State	Description	ModeTypeEnum Software Constant
Embedded	The class behavior definition is created, saved to a ClassBehavior schema file, and is embedded in the iXF archive file. The ClassBehavior schema is specified by the namespace of the behavior.	mtEmbedded
External	<p>The class behavior was previously defined and the definition is in a schema file. The ClassBehavior definition is taken from the external ClassBehavior schema file.</p> <p>The ClassBehavior schema file is not embedded in the iXF package; it is specified by its BehaviorURI.</p>	mtExternal

Add Method

The Add method is called as follows:

```
Set ClassBehavior = Schema.ClassesBehaviors.Add(ModeTypeEnum, BehaviorURI,  
[SchemaLocation], [Load=false])
```

The arguments of the method are:

Argument	Description
Mode	Packaging state of the schema – one of ModeTypeEnum. See table above.
BehaviorURI	The Error! Not a valid result for table.
SchemaLocation	The behavior schema physical location. Specified only when the Mode is mtExternal.
Load	Whether or not the behavior schema needs to be loaded. The default is false. Can be set to true only when the Mode argument is mtExternal.

Example

```
Dim IxfClassBehavior as ISmIxfClassBehavior  
  
'Create a Class Behavior "link"
```

```
Set IxfClassBehavior = Schema.ClassesBehaviors.Add(  
    mtEmbedded,  
    "http://www.ixfstd.org/std/ns/core/classBehaviors/links/1.0#link")
```

See ISmIxfClassBehavior for more information on that object.

ISmIxfClassBehavior

A ClassBehavior lets you define a set of class attributes as an entity separate from any specific class, where the entity is identified by a unique URI reference. A ClassBehavior so defined becomes a standard set of attributes, which can be “implemented” by one or more classes, as required. In this way, a ClassBehavior is similar to an Interface of a programming language like Java™; any class that wants to implement a ClassBehavior needs to declare the ClassBehavior usage.

A ClassBehavior is defined in the schema separately from the class definitions and is used in a specific class by declaring it in the class definition in the schema (see Adding a ClassBehavior to a Class in ISmIxfClassBehaviors.) The attribute values for attributes in the ClassBehavior are assigned in the instantiation of the class in the data file, in the same way that values are assigned to the internal attributes of the class.

Figure 10-6 shows how ClassBehaviors are used in schema class definitions. It shows the internal attribute definitions of a class and the ClassBehavior declarations. You can declare the same ClassBehavior in more than one class, and you can declare more than one ClassBehavior in a single class.

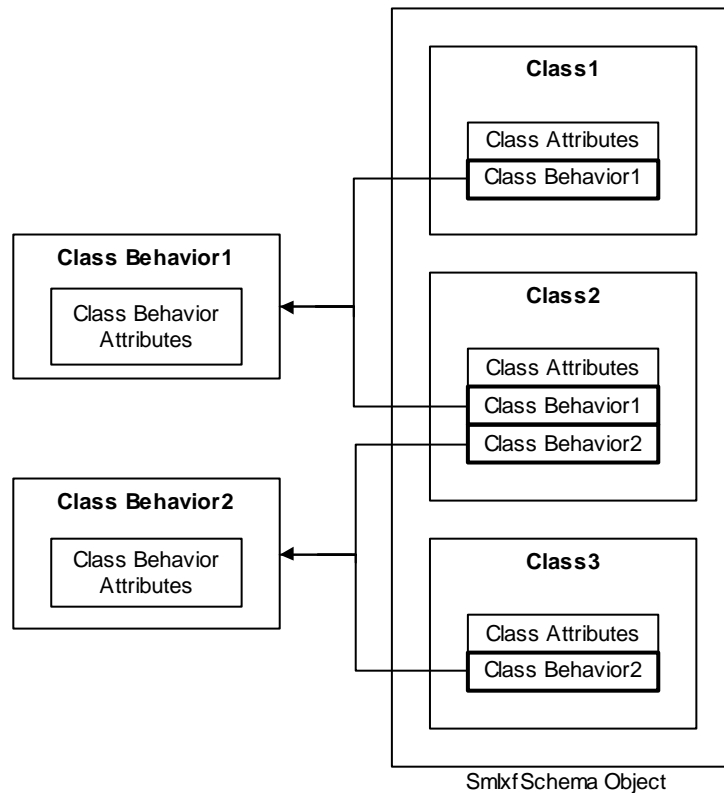


Figure 11-2 Class Behaviors

Properties

An ISmIxfClassBehavior object has the following properties:

Property	Description
Attributes	Collection ISmIxfAttributes of class behavior attributes
SchemaLocation	The physical location of the ClassBehavior schema file, in the event that the ClassBehavior was previously defined and saved to a schema file, definition can be obtained (loaded) through this schema file. Otherwise the class behavior is defined in the schema (and not loaded) – the SchemaLocation parameter is an empty string.
URI	The Error! Not a valid result for table., which is the unique identifier the class behavior.

Adding an Attribute to a ClassBehavior

Use the Add method of the IxfClassBehavior.Attributes collection object to add an attribute to the IxfClassBehavior.

```
Set IxfAttribute = IxfClassBehavior.Attributes.Add(AttributeName)
```

The Add method returns an object of type ISmIxfAttribute. Specify the AttributeName as a valid NCName. For more information about Attributes, see ISmIxfAttribute.

For an example of how to add an attribute to a ClassBehavior, see Common Tasks, “ISmIxfSchema: Defining a ClassBehavior”.

ISmIxfClasses

An ISmIxfClasses object is a collection of ISmIxfClass objects.

Adding a Class to the IXF Schema

Use the Add method of the ISmIxfClasses object to add a class to the IXF Schema. The Add method returns an object of type ISmIxfClass. You specify the name of the class as a valid NCName.

The Add method is called as follows:

```
Set IxfClass = Schema.Classes.Add(ClassName)
```

Where ClassName has to be a valid NCName.

Once you have added the class you can specify the class properties, as in the next section.

For an example of how to add a class to the IXF Schema, see Common Tasks, “ISmIxfSchema: Creating a Schema with Classes and Class Attributes”.

ISmIxfClass

The following figure shows the object diagram for the ISmIxfClass Object.

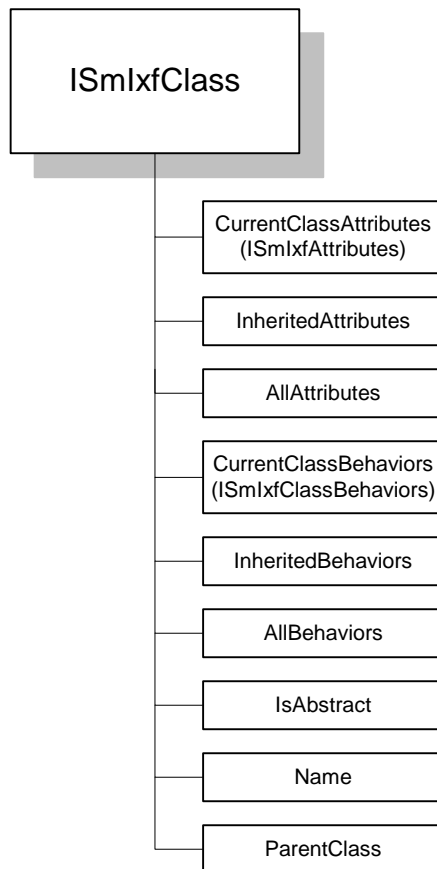


Figure 11-3 ISmIxfClass Object Diagram

Properties

An ISmIxfClass object has the following properties:

Property	Description
Name	The class name must be a valid NCName.
Parent Class	The parent class object
CurrentClassAttributes	Collection ISmIxfAttributes of the class attributes. Does not include the inherited class attributes.
InheritedAttributes	Collection ISmIxfReadOnlyAttributes of the inherited class attributes.
AllAttributes	Collection ISmIxfReadOnlyAttributes of all the class attributes, including inherited attributes.
CurrentClassBehaviors	Collection ISmIxfClassBehaviors of class behaviors that are supported by the class. Does not include the class behaviors that were inherited.

InheritedBehaviors	Collection ISmIxfReadOnlyClassBehaviors of the inherited class behaviors.
AllBehaviors	Collection ISmIxfReadOnlyClassBehaviors of all the class behaviors that are supported by the class, including inherited class behaviors.
IsAbstract	Indicates whether or not the class is abstract. If it is, it cannot be instantiated

ISmIxfAttributes

An ISmIxfAttributes object is a collection of ISmIxfAttribute objects.

The CurrentClassAttributes is a collection of ISmIxfAttributes objects, which represent the attributes defined internally to the current class. The ClassBehavior object also includes a collection ISmIxfAttributes, which represents the attributes of the ClassBehavior (see ISmIxfClassBehaviors.)

Adding an Attribute to a Class

Use the Add method of the ISmIxfAttributes object to add an attribute to the collection. The Add method returns an object of type ISmIxfAttribute. It is called as follows:

```
Set IxfAttribute = IxfClass.CurrentClassAttributes.Add(AttributeName)
```

Where AttributeName must be a valid NCName.

Once you have added the attribute you can specify the attribute properties.

For an example of how to add an attribute to CurrentClassAttributes, see Common Tasks, “ISmIxfSchema: Creating a Schema with Classes and Class Attributes”.

ISmIxfAttribute

The ISmIxfAttribute object represents an individual class attribute or an individual attribute of a ClassBehavior.

Properties

The ISmIxfAttribute object has the following properties:

Property	Description
----------	-------------

Name	The attribute name. A valid NCName.
Default value	The default value of the attribute. It is assigned as an object's (class or ClassBehavior) attribute value in case no value was assigned.
IsNullAllowed	True if the attribute value can be set to Null. Default is true
IsPrimary	True if the attribute is part of the class primary identifier. The default is false.
Required	True if the attribute is required. If it is, it has to be assigned, or a default value must be indicated in the Default value property. Default is false.
TypeDefinition	Returns the data type of the attribute. Returns an object of type ISmIxfTypeDefinition.

Note: ISmIxfAttribute is only the definition of the attribute structure; the actual value of this attribute is inserted by the SmlxfWriter object.

ISmIxfTypeDefinition

The ISmIxfTypeDefinition Object specifies the data type of the ISmIxfAttribute object.

Properties

The ISmIxfTypeDefinition object has three properties:

Property	Description
ValueType	The ValueType property is an Enum type DataTypeEnum that specifies the type of the value that can be assigned to the attribute. It is a subset of the W3C XML Schema Data Types, as defined in http://www.w3.org/TR/xmlschema-2/ . See Table 5 for a list of data types.
ObjectReferenceType	If ValueType is set to dtObjectReference, the ObjectReferenceType property lets you specify more information about the object reference. See below for more information.
StringType	If the ValueType is assigned to dtString then the StringType property lets you specify more information about the string. See below for more details.

Table 5 ValueType Data Types

ValueType	Description	Software Constant
String	Character strings in XML	dtString
Boolean	Binary-valued logic	dtBoolean
Float	IEEE single-precision 32-bit floating point type	dtFloat
Double	IEEE double-precision 64-bit floating point type	dtDouble
Duration	Duration of time	dtDuration
Base64Binary	Base64-encoded arbitrary binary data	dtBase64Binary
HexBinary	Arbitrary hex-encoded binary data	dtHexBinary
AnyUri	A Uniform Resource Identifier Reference (URI)	dtAnyUri
Language	Natural language identifiers as defined by [1766] .	dtLanguage
Int	Integer between -2147483648 and 2147483647.	dtInt
Short	Integer between -32768 and 32767	dtShort
Byte	Integer between -128 and 127	dtByte
UnsignedShort	Integer between 0 and 65535	dtUnsignedShort
UnsignedByte	Integer between 0 and 255	dtUnsignedByte
DateTime	A specific instant of time	dtDateTime
Time	An instant of time that recurs every day	dtTime
Date	A calendar date	dtDate
gMonth	A gregorian month that recurs every year	dtGMonth
gYear	A gregorian calendar year	dtGYear
ObjectReference	An object	dtObjectReference
XML	XML text	dtXML

Depending on ValueType, there can be additional options, as discussed in the following sections.

Specifying Information about an Object Reference

If `ISmIxfTypeDefinition.ValueType` is set to `dtObjectReference`, the `ObjectReferenceType` property lets you specify more information about the object reference.

Properties

The `ObjectReferenceType` property returns an object of type `ISmIxfObjectReferenceTypeDefinition`, which has the properties:

Property	Description
RestrictionType	You use this property to place restrictions on the type of object referenced through the ObjectReferenceType property. Set to one of the values in the ObjectReferenceRestrictionTypeEnum .
ClassName	Specifies a class to which the objects referenced or their descendants must belong. Can be accessed only if RestrictionType has the value ortClass or ortClassAndDescendants. See details of ObjectReferenceType.RestrictioType below for more information.
BehaviorURI	Specifies a behavior that the object referenced must implement. Can be accessed only if the RestrictionType has the value ortBehavior. See details of ObjectReferenceType.RestrictioType below for more information.

ObjectReferenceType Restrictions

You use the RestrictionType property to place restrictions on the type of object that can be referenced through the ObjectReferenceType property. This helps you tailor an attribute for special use.

The RestrictionType property is available as follows:

```
IxfAttribute.TypeDefinition.ObjectReferenceType.RestrictioType
```

The `RestrictionType` can take one of the following `ObjectReferenceRestrictionTypeEnum` values:

RestrictionType	Description	Software Constant
Any	The reference can be to any kind of object (the default)	<code>ortAny</code>
Class	The reference can be to an object of a specific class only. The class name should be assigned to <code>TypeDefinition.ObjectReferenceType.ClassName</code> .	<code>ortClass</code>
ClassAnd Descendants	The reference can be to an object of a specific class or its descendants only. The class name should be assigned to <code>TypeDefinition.ObjectReferenceType.ClassName</code> .	<code>OrtClassAnd Descendants</code>
Behavior	The reference can be to an object that implements specific behavior only. The behavior URI should be assigned to <code>TypeDefinition.ObjectReferenceType.BehaviorURI</code>	<code>ortBehavior</code>

Example

The following code allows the attribute to reference only objects of the class “DocumentMaster”.

```

IxfAttribute.TypeDefinition.ValueType = dtObjectReference

IxfAttribute.ObjectReferenceType.RestrictionType = ortClass

IxfAttribute.ObjectReference.ClassName = "DocumentMaster"

```

String Type Options

If the `ISmIxfTypeDefinition.ValueType` is assigned to `dtString` then you can specify a maximum length for the string by `ISmIxfTypeDefinition.StringType.MaxLength`. The default for `MaxLength` is 0, which means there is no restriction for the string length.

Example

The following code restricts the length of the string attribute to 50 characters.

```
IxfAttribute.TypeDefinition.ValueType = dtString  
  
IxfAttribute.TypeDefinition.StringType.MaxLength = 50
```

ISmIxfClassBehaviors

The `ISmIxfClassBehaviors` Object is a collection of `ISmIxfClassBehavior` objects. It represents the set of `ClassBehaviors` declared in a specific class.

Note: This object is not the same as the collection object `ISmIxfClassesBehaviors`. The latter refers to the set of all `ClassBehavior` objects associated with the entire schema and defined externally to all classes.

Adding a ClassBehavior to a Class

Once you have added a `ClassBehavior` to the `ClassesBehaviors` object (see [Adding a New ClassBehavior to the IXF Schema](#)), you can declare the `ClassBehavior` in a specific class. You use the `Add` method of the `ClassBehaviors` object to add a `ClassBehavior` object to `CurrentClassBehaviors`.

The `Add` function is called as follows:

```
IxfClass.CurrentClassBehaviors.Add(Behavior, MustUnderstandEnum)
```

The method parameters are as follows:

Parameter	Description
Behavior	A ClassBehavior object that already exists in the collection Schema.ClassesBehaviors.
MustUnderstand	Denotes whether this Class Behavior, when declared in this class, must be understood by the reading processor. Possible values are muYes, muNo

For an example, see Common Tasks, ISmIxfSchema: Declaring usage of a class behavior by a class.

ISmIxfDomainBehaviors

An ISmIxfDomainBehaviors object is a collection of ISmIxfDomainBehavior objects.

Adding a DomainBehavior to DomainBehaviors

You use the Add method of the DomainBehaviors object to add a DomainBehavior object to DomainBehaviors.

The Add function is called as follows:

```
Add(URI)
```

ISmIxfDomainBehavior

Conceptually, a Domain Behavior is composed of sets of Class Behaviors called Roles, where the Domain Behavior also specifies the classes that declare the Class Behaviors for each Role.

Specifically, a Domain Behavior defines a set of Roles and a set of Role-to-Class mappings (see Section 2.5 of the IXF Specification.) Each Role is associated with a set of Class Behaviors, which are specified by the documentation describing the Domain Behavior. The class that is mapped to the Role according to its Role-to-Class Mapping must declare the Role's Class Behaviors.

Note: It is very important to make sure that a class is mapped to a Role only if it implements the required class behaviors, even though this is not currently enforced by the API. The required class behaviors can be verified by consulting the Domain Behavior documentation.

Figure 11-4 shows the relationship between a Domain Behavior definition and a schema that uses it:

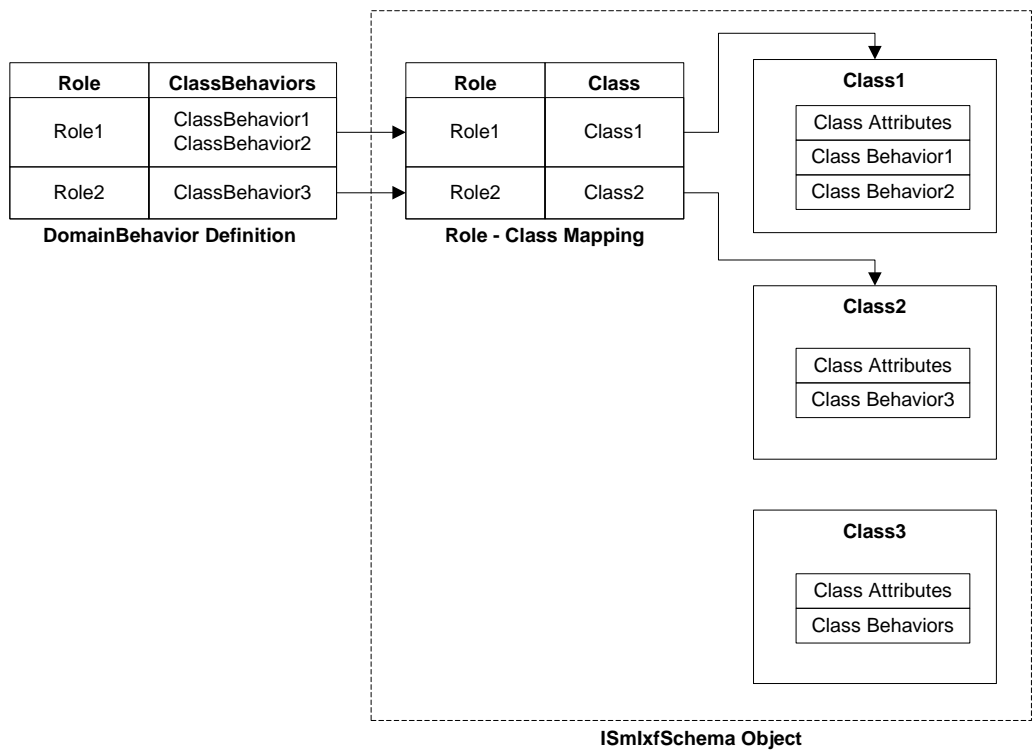


Figure 11-4 Domain Behavior

Properties

An ISmIxfDomainBehavior object has the following properties:

Property	Description
URI	The unique identifier of the DomainBehavior
RoleClassMapping	The RoleClassMapping property defines an association between names and ISmIxfClass objects, whereby each role is assigned to a class.

ISmIxfInfo

The ISmIxfInfo Object holds miscellaneous information, which cannot be categorized as classes or behaviors. It is a collection of ISmIxfInfoItem objects.

Methods

It has the methods:

Method	Description
GetInfoItem	Gets an InfoItem from the collection by Name and Namespace. If it doesn't exist, a new object is created and added to the collection.
Save	Not used when Info accessed through Schema.
CreateXmlAttributeValue	Creates an ISmIxfXmlAttributeValue object

The ISmIxfInfo Object is obtained through the ISmIxfSchema object, as follows:

```
Set IxfInfo = Schema.Info
```

Note: ISmIxfInfo Object can also be obtained through the IxfWriter.DataWriter and IxfReader.DataReader. When the schema is saved ISmIxfInfo is saved automatically with the schema; in the Writer it has to be saved with the Save function.

Adding an InfoItem to a Info Object

Use the GetInfoItem method of Info to add a new InfoItem to the Info collection.

```
Set IxfInfoItem = IxfInfo.GetInfoItem(Name, Namespace)
```

ISmIxfInfoItem

The ISmIxfInfoItem object represents a member of the ISmIxfInfo collection, that is, a basic unit of miscellaneous information in the schema file.

Properties

The ISmIxfInfoItem object has the properties:

Property	Description
Name	InfoItem name
Namespace	InfoItem namespace
Value	InfoItem value
ValueType	Value type (see Table 5)
MustUnderstand	MustUnderstand flag for this InfoItem. If set to true, the reading process stops when this InfoItem is not in the list SmlxfReader.ISmIxfUnderstoodInfoItems, and Reader.ValidateMustUnderstand = true.

Use the GetInfoItem (Name, Namespace) method of the ISmIxfInfo Object to create an ISmIxfInfoItem object or to get an existing one.

For an example of how to add miscellaneous information to a ClassBehavior, see Common Tasks, ISmIxfSchema: Adding miscellaneous information.

Note: The ISmIxfInfo object under DataWriter represents another, independent way to write miscellaneous data, which you can use instead of or in addition to this ISmIxfInfo object under the ISmIxfSchema object. The difference is that with the current ISmIxfInfo object you do not need to save the object; it is saved automatically with the Schema.

ISmIxfXmlAttributeValue

The ISmIxfXmlAttributeValue object represents the value of an InfoItem of type “dtXML”. This object lets you insert miscellaneous information in the form of XML text. The XML text does not need to be a complete XML document, but it must be valid and well formed.

If the meaning of a prefix is not included in the XML text, you can provide it in the Namespaces property.

To create an ISmIxfXmlAttributeValue object, use the ISmIxfInfo method CreateXmlAttributeValue, as follows:

```
Set IxfXmlAttributeValue = Info.CreateXmlAttributeValue
```

Properties

The ISmIxfXmlAttributeValue object has the properties:

Property	Description
Namespaces	Returns an object ISmIxfNamespaces, a list of mappings between prefix and namespace that represents the meaning of each prefix that occurs in the XML string.
XML	A well-formed valid XML text as a WideString.

Note: The ISmIxfXmlAttributeValue object can also be used to provide an XML text attribute value to a class attribute, when using the Writer object. See ISmIxfAttributesValues for more information.

Note: The XML string might be changed by the API but the meaning will stay the same.

Example

```
Dim Info As ISmIxfInfo
Dim InfoItem As ISmIxfInfoItem
Dim NameSpaces As ISmIxfNamespaces
Dim XmlAttributeValue As ISmIxfXmlAttributeValue
Dim XmlText As String

Set Info = Schema.Info
InfoItem = Info.GetInfoItem("XmlText", "http://...")
InfoItem.ValueType = dtXml
Set XmlAttributeValue = Info.CreateXmlAttributeValue
XmlText = "<p:name>John Bryce<p:name>"
XmlAttributeValue.XML = XmlText
XmlAttributeValue.Namespaces.Add ("ns1", "prefix1")
InfoItem.Value = XmlAttributeValue
```

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to a ISmIxfSchema.

ISmIxfSchema: Creating a Schema with Classes and Class Attributes

The following procedure creates a basic schema file containing classes and class attributes.

Get the schema property (see ISmIxfSchema.)

Create a class

```
Dim IxfClass as ISmIxfClass
```

```
'Create a Class "documentMaster":
```

```
Set IxfClass = Schema.Classes.Add("DocumentMaster")
```

```
IxfClass.ParentClass = Null
```

```
IxfClass.IsAbstract = False
```

See Adding a Class to the IXF Schema, for more details.

Create a Class attribute

```
Dim IxfAttribute as ISmIxfAttribute
```

```
'Create and add attribute "DocumentName":
```

```
Set IxfAttribute = IxfClass.CurrentClassAttributes.Add("DocumentName")
```

```
'Set properties for the attribute:
```

```
IxfAttribute.TypeDefinition.ValueType = dtString
```

```
IxfAttribute.TypeDefinition.StringType.MaxLength = 50
```

```
IxfAttribute.Required = True
```

```
IxfAttribute.IsNullAllowed = False
```

```
IxfAttribute.IsPrimary = True
```

```
'Create and add attribute "Description":
```

```

Set IxfAttribute = IxfClass.CurrentClassAttributes.Add("Description")

'Set properties for the attribute:

IxfAttribute.TypeDefinition.ValueType = dtString

IxfAttribute.Required = False

IxfAttribute.IsNullable = True

```

See Adding an Attribute to a Class for more information.

ISmIxfSchema: Defining a ClassBehavior

In this task, you define a ClassBehavior.

Add a Class Behavior definition to the IXF Schema, that is, to
ClassesBehaviors

```

Dim IxfClassBehavior as ISmIxfClassBehavior

'Create a Class Behavior "link"

Set IxfClassBehavior = Schema.ClassesBehaviors.Add(
    mtEmbedded,
    "http://www.ixfstd.org/std/ns/core/classBehaviors/links/1.0#link")

```

See Adding a New ClassBehavior, for more information.

Add attributes to the ClassBehavior definition

```

Dim IxfAttribute as ISmIxfAttribute

'add attribute "object1":

Set IxfAttribute = IxfClassBehavior.Attributes.Add("object1")

'Set properties for the attribute:

IxfAttribute.TypeDefinition.ValueType = dtObjectReference

IxfAttribute.Required = True

IxfAttribute.IsNullable = True

'add attribute "object2":

Set IxfAttribute = IxfClassBehavior.Attributes.Add("object")

```

'Set properties for the attribute:

```
IxfAttribute.TypeDefinition.ValueType = dtObjectReference
```

```
IxfAttribute.Required = True
```

```
IxfAttribute.IsNullable = True
```

See [ISmManagedClasses](#). The ISmManagedClasses object represents the set of SmarTeam managed classes to which a specific integration behavior is mapped for more information.

ISmIxfSchema:

Declaring usage of a Class Behavior by a class

In this task, you declare a ClassBehavior in a class.

In case the ClassBehavior has been defined separately, you can retrieve the class behavior object by URI and declare it in a class as follows:

```
IxfClassBehavior = Schema.classesBehaviors.ItemByURI(  
    "http://www.ixfstd.org/std/ns/core/classBehaviors/links/1.0#link")  
  
IxfClass = Schema.classes.Add("myLink")  
  
IxfClass.CurrentClassBehaviors.Add IxfClassBehavior
```

See Adding a ClassBehavior to a Class for more information.

ISmIxfSchema:

Defining a DomainBehavior

See example section.

ISmIxfSchema:

Adding Standard Behavior to a schema

See section [ISmIxfSchemaHelper](#) on page [403](#)

ISmIxfSchema:

Adding miscellaneous information

To add miscellaneous information ISmIxfInfo to the schema, you use an ISmIxfInfoItem object

```
Dim IxfInfo As ISmIxfInfo
```

```
Dim IxfInfoItem As ISmIxfInfoItem
```

```
' Get the Info object
Set IxfInfo = Schema.Info

'Create and return an InfoItem with the indicated name and namespace:
Set IxfInfoItem = IxfInfo.GetInfoItem("transaction", "http://www.vendor.org")

IxfInfoItem.ValueType = dtInt
IxfInfoItem.Value = "2352"
```

SmIxfInitializationData

A **SmIxfInitializationData** object represents initialization of data for SmartIXF applications.

Each creatable object has a reference to the interface **ISmIxfInitializationData**.

Setting Proxy Information

For the present release, the **SmIxfInitializationData** interface relates to the initialization of proxy information for downloading files by an IXF Application installed on a UNIX system (optional on Windows).

The user can obtain the current proxy value by calling the **GetProxy** method.

The **SetProxy** method should be performed when the user wants to indicate the proxy that is about to be used.

On Unix platforms the proxy must be set if files are about to be downloaded from the web.

On windows using this interface is optional since windows can automatically detect and identify a proxy.

Methods

The **SmIxfInitializationData** has the methods

Methods	Description
GetProxy	Returns the value of the proxy string.
SetProxy	Sets the specified string as the proxy string.

Example

In order to set a proxy after creating an object, the SetProxy method should be called as described in the following sample code.

```
Public Const PROXY_STR = "123.45.678.90:8080" 'A string indicating the proxy
to be used when downloading files from the web.
```

```
Dim IxfWriter As SmIxfWriter
```

```
Dim IxfReader As SmIxfReader
```

```
Dim IxfStdHelper As SmIxfStdHelper
```

```
Dim IxfExternalSchemaWriter As SmIxfExternalSchemaWriter
```

```
Dim IxfExternalSchemaReader As SmIxfExternalSchemaReader
```

```
Dim ProxyStr As String
```

```
Set IxfWriter = CreateObject("SmartIXF1.SmIxfWriter")
```

```
IxfWriter.InitializationData.SetProxy(PROXY_STR)
```

```
Set IxfReader = CreateObject("SmartIXF1.SmIxfReader")
```

```
IxfReader.InitializationData.SetProxy(PROXY_STR)
```

```
Set IxfStdHelper = CreateObject("SmartIXF1.SmIxfStdHelper")
```

```
IxfStdHelper.InitializationData.SetProxy(PROXY_STR)
```

```
Set IxfExternalSchemaWriter = CreateObject("SmartIXF1.ExternalSchemaWriter")
```

```
IxfExternalSchemaWriter.InitializationData.SetProxy(PROXY_STR)
```

```
Set IxfExternalSchemaReader = CreateObject("SmartIXF1.ExternalSchemaReader")
```

```
IxfExternalSchemaReader.InitializationData.SetProxy(PROXY_STR)
```

```
...
```

```
ProxyStr = IxfReader.InitializationData.GetProxy()
```

SmIxfWriter

An **SmIxfWriter** object is used for:

- creating an IXF Archive file
- creating and writing a Schema Document (schema file)
- creating and writing an IXF Instance Document (data file)
- packaging the schema and data files in the IXF Archive file.
- Optionally, the SmIxfWriter can refer to an existing schema file.

Object Diagram

The object diagram of SmIxfWriter is shown below:

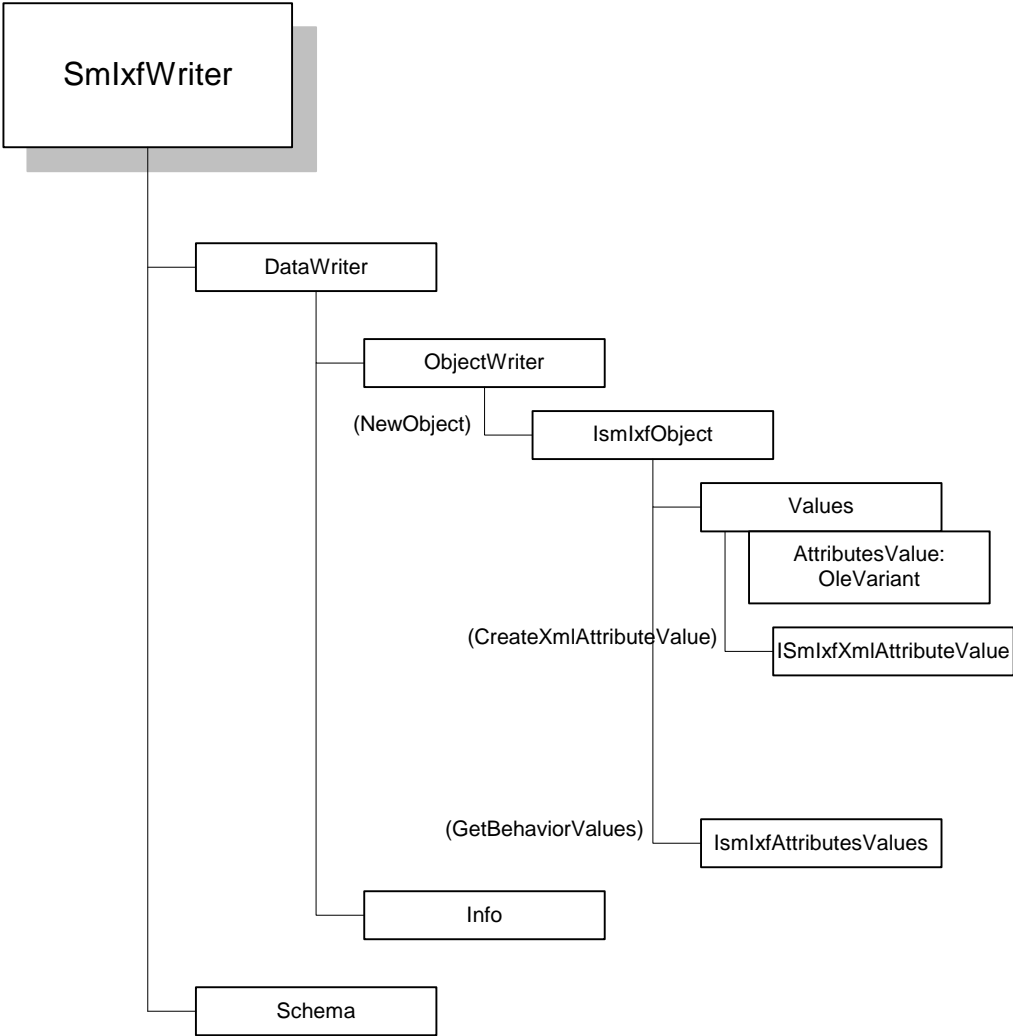


Figure 11-5 SmlxfWriter Object Diagram

Properties

SmIxfWriter has the following properties:

Property	Description
Schema	Holds the definition of the data structure. The schema file describes the data model, including its classes and behaviors.
DataWriter	Writes the data file. The data file contains a set of objects that conforms to the data model described in the schema and miscellaneous data. [associated files come from WriterHelper]
InitializationData	Provides access to methods for initializing data for IXF application. Returns ISmIxfInitializationData

The schema and data files are packaged in an IXF Archive by the SmIxfWriter object.

Methods

The SmIxfWriter has the methods

Methods	Description
CreateIxfArchiveFile	Creates the specified iXF Archive file.
CloseIxfArchiveFile	Closes the iXF Archive file.
SetSchemaMode	Sets the packaging mode of the schema in the Archive file.

Creating the SmIxfWriter Object

To create an SmIxfWriter Object:

```
Dim IxfWriter As SmIxfWriter  
  
Set IxfWriter = CreateObject("SmartIXF1.SmIxfWriter")
```

Creating an iXF Archive File

As described in the iXF Specification, an iXF Archive file is a zip file containing a data file and possibly a schema file. When you use the SmIxfWriter to write an IXF Instance file, you need to create an iXF Archive file to contain the IXF Instance file and possibly the schema file.

Specifying the Schema Packaging State

When you create an iXF Archive file, you first need to specify how the associated schema is to be packaged, using the SetSchemaMode method of the ISmIxfWriter.

The following table describes the packaging states of the schema file and the software constant used for each state. The packaging state of the schema file is described by the Mode parameter of the SetSchemaMode method.

Packaging State	Description	SchemaModeEnum Software Constant
Embedded	The schema definition is created and saved to a schema file, which is embedded in the iXF archive file.	mtEmbedded
External	The schema file is not embedded in the iXF package; it was previously defined externally in a file and specified by its SchemaURI.	mtExternal

SetSchemaMode Method

Use the SetSchemaMode method to specify the packaging state of the schema. It is called as follows:

```
IxfWriter.SetSchemaMode(mode,[SchemaURI],[SchemaLocation = ""] [Load=True])
```

The arguments of the method are:

Argument	Description
Mode	Packaging state of the schema, one of SchemaModeEnum. See table above.
SchemaURI	The schema namespace. Specified only when Mode is mtExternal.
SchemaLocation	The schema physical location. Specified only when Mode is mtExternal.
Load	Whether or not the schema needs to be loaded. The default is false. Can be set True only when Mode is mtExternal.

Note: If the Mode is mtEmbedded, or if Mode is mtExternal and Load = False (for example, when the schema is not accessible) then the Schema object in the ixfWriter object should be populated by hand. See ISmIxfSchema.

Creating the IXF Archive

To create the iXF Archive, after calling the SetSchemaMode method, use the methods IxfWriter.CreateIxfArchiveFile and IxfWriter.CloseIxfArchiveFile.

For an example of how to create the IXF Archive, see Common Tasks, SmIxfWriter: Creating an iXF Archive.

ISmIxfDataWriter

The ISmIxfDataWriter Object writes the object and miscellaneous data corresponding to the schema file.

Properties

The ISmIxfDataWriter Object has the two properties:

Property	Description
ObjectWriter	Returns an ISmIxfObjectWriter object, used to write objects to data file.
Info	Returns an ISmIxfInfo object. It is used for writing miscellaneous information to the data file.

Obtaining the ISmIxfDataWriter Object

To obtain the ISmIxfDataWriter Object from the IxfWriter Object:

```
Dim DataWriter as ISmIxfDataWriter  
Set DataWriter = IxfWriter.DataWriter
```

ISmIxfObjectWriter

The ISmIxfObjectWriter uses the NewObject method to create objects, which are instantiations of the classes declared in the schema.

Obtaining the ISmIxfObjectWriter Object

To obtain the ISmIxfObjectWriter object from the ISmIxfDataWriter object:

```
Dim ObjectWriter as ISmIxfObjectWriter  
Set ObjectWriter = IxfWriter.DataWriter.ObjectWriter
```

Creating a New Object

Use the NewObject method to create an object, which is an instantiation of a class defined in the Schema. Use the Class Name and provide a unique Object Id (see next section for more details about the parameters).

```
Set IxfObject = ObjectWriter.NewObject(ixfClassName, ObjectId)
```

ISmIxfObject

The ISmIxfObject object represents an instantiation of a class in the schema. You create it a ISmIxfObject object by specifying the class from which the object is to be instantiated and providing an object id (see previous section).

Use the ISmIxfObject object to access class attributes and class behavior attributes that were declared in the schema file for the object's class.

The ISmIxfObject has the properties:

Property	Description
Id	Input string that uniquely identifies the object within the IXF Instance Document. Must be a valid NCName. The Object ID value must follow the rules defined for the ID Datatype in XML Schema Part 2: Datatypes, Section 3.3.8: ID.
IxfClassName	Name of class of which this object is an instantiation.
Values	Returns object ISmIxfAttributesValues, which is the set of values the class attributes for the object's class.

The ISmIxfObject has the methods:

Method	Description
GetBehaviorValues	Returns object ISmIxfAttributesValues, which is the set of the ClassBehavior attribute values for Class Behaviors declared by object's class.
Save	Saves object to data file. Can be used only during iXF generation (writing)

ISmIxfAttributesValues

The ISmIxfAttributesValues object, which is returned by the Values and GetBehaviorValues methods of an IxfObject, represents the collection of values of class attributes or ClassBehavior attributes of the IxfObject. You refer to an individual item of the ISmIxfAttributesValues by the name of the corresponding ISmIxfAttribute, which was assigned in the class or ClassBehavior definition in the schema.

```
IxfObject.Values.Item(AttributeName) = some variant value
```

```
Set BehaviorValues = IxfObject.GetBehaviorValues(BehaviorURI)
```

```
BehaviorValues.Item(AttributeName) = some variant value
```

For an example of how to assign values to class attributes, see Common Tasks, ISmIxfDataWriter: Creating a Data File with Objects and Info.

ISmIxfXmlAttributeValue Object

If you defined the TypeDefinition.ValueType of a class attribute or ClassBehavior attribute as dtXml in the schema definition, you can create an ISmIxfXmlAttributeValue object and assign it as the class attribute value.

```
Set XmlAttributeValue = IxfObject.Values.CreateXmlAttributeValue
```

Note: The ISmIxfXmlAttributeValue object can also be used when using the ISmIxfInfo object to provide an Xml text value to an Infoltem of type dtXml.

Example

'In definition of ClassAttributes in Schema, define an Xml attribute:

```
Set IxfAttribute = IxfClass.CurrentClassAttributes.Add("XmlText")
```

'Set properties for the attribute:


```

IxfAttribute.TypeDefinition.ValueType = dtXml

---

'When loading values into Class attributes in Writer:

Dim DataWriter As ISmIxfDataWriter
Dim ObjectWriter As ISmIxfObjectWriter
Dim Object As ISmIxfObject
Dim XmlAttributeValue As ISmIxf XmlAttributeValue
Dim XmlText As String

Set Object = DataWriter.ObjectWriter.Object

'Create XmlAttributeValue object and give it an Xml value
XmlAttributeValue = Object.Values.CreateXmlAttributeValue
XmlText = "<p:name>John Bryce<p:name>"
XmlAttributeValue.XML = XmlText
XmlAttributeValue.NameSpaces.Add ("p", "prefix1")

'Put the XmlAttributeValue object into the class attribute value
Object.Value["XmlText"] = XmlAttributeValue

Object.Save

```

ISmIxfInfo

The ISmIxfInfo object represents miscellaneous information written to the data file. See the ISmIxfInfo object of the Schema object.

The ISmIxfInfo Object holds miscellaneous information, which cannot be categorized as classes or behaviors. It is a collection of ISmIxfInfoItem objects.

Methods

It has the methods:

Method	Description
--------	-------------

GetInfoltem	Gets an Infoltem from the collection by Name and Namespace.
Save	Saves the Infoltems collection to the iXF Instance file. Can be used only when Info is obtained through IxfWriter.DataWriter, i.e., during iXF generation.
CreateXmlAttributeValue	Creates an ISmIxfXmlAttributeValue object

Note: This ISmIxfInfo object under DataWriter represents an independent way to write miscellaneous data, which you can use instead of or in addition to the ISmIxfInfo object under the ISmIxfSchema object. The difference is that with this ISmIxfInfo object you need to save the object, as shown below.

For an example of how to write an Info object, see Common Tasks, ISmIxfDataWriter: Writing an Info section.

Note: The ISmIxfInfo information must be written to the data file prior to any object information.

ISmIxfSchema

The ISmIxfSchema object represents the schema file in the IXF Archive being written. See ISmIxfSchema on page 356.

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to a SmIxfWriter.

SmIxfWriter: Creating an iXF Archive

As described in the iXF Specification, an iXF Archive file is a zip file containing a data file and possibly a schema file.

To create an iXF package file:

Create the IxfWriter object

```
Dim IxfWriter As SmIxfWriter

Set IxfWriter = CreateObject("SmartIXF1.SmIxfWriter")
```

Set the schema packaging state using the method SetSchemaMode:

Examples of setting the schema packaging mode:

```
IxfWriter.SetSchemaMode mtEmbedded
```

or:

```
IxfWriter.SetSchemaMode mtExternal,
"http://www.vendor.org.schema",
"c:\Schemas\MySchema.xsd", true
```

or :

```
IxfWriter.SetSchemaMode mtExternal,
"http://www.vendor.org.schema",
"c:\Temp\MySchema.xsd", False
```

See Creating an iXF Archive File, for more information.

Create a schema (see ISmIxfSchema)

If the ModeTypeEnum is embedded, or if ModeTypeEnum is external and Load = False (for example, when the schema is not accessible) then the Schema object in the ixfWriter object should be populated by hand.

Use the method CreateIxfArchiveFile to initialize the process of creating an IXF Archive:

```
IxfWriter.CreateIxfArchiveFile "test.ixf"
```

Note: This method can be called only after SetSchemaMode is called and the schema object is populated.

Insert the data information -- Info, objects, changes and files (see next section "Creating a Data File with Objects and Info")

Close the iXF file:

```
IxfWriter.CloseIxfArchiveFile
```

ISmIxfDataWriter: Creating a Data File with Objects and Info

The following procedure creates a basic data file containing objects and miscellaneous information. It is assumed that a schema has already been created.

The data file is created automatically as part of the package. It is named IXF_Data.xml.

In this section you create an object corresponding to a class in the schema and assign values to the class attributes and ClassBehavior attributes for class behaviors declared by the class.

Note: If you are writing miscellaneous (Info) information, it must be written first, before any object information.

Create an object

To create an object, you need to obtain the ISmIxfObjectWriter Object as follows:

```
Dim ObjectWriter As ISmIxfObjectWriter
```

```
Set ObjectWriter = IxfWriter.DataWriter.ObjectWriter
```

Use the `NewObject` method of the `ISmIxfObjectWriter` Object to create the new object, where you specify the `ClassName` and provide a unique `ObjectId`. The `NewObject` method returns an object of type `ISmIxfObject`.

```
' Create an object for the data file
```

```
Dim ixfObject as ISmIxfObject
```

```
Set ixfObject = ObjectWriter.NewObject("DocumentMaster", "OID_1")
```

Assign values according to the object's class attributes

The attribute values are stored in the collection object `ISmIxfAttributesValues`. This object is obtained from `ISmIxfObject` as follows:

```
AttributesValues = ixfObject.Values
```

You assign a value to this object.

```
' Assign values to the object's attributes
```

```
AttributesValues.Item("DocumentName") = "MyDocument"
```

Assign values to the `ClassBehavior` attributes

To assign values to the `ClassBehavior` attributes, use the method `GetBehaviorValues` of `ISmIxfObject`. The method returns the object `ISmIxfAttributesValues` as in the previous step.

```
Dim BehaviorValues As ISmIxfAttributesValues
```

```
IxfLinkObject = IxfWriter.DataWriter.ObjectWriter.NewObject("MyLink", "OID_2")
```

```
Set BehaviorValues =
```

```
IxfLinkObject.GetBehaviorValues(http://project/behavior1#link)
```

You assign values to this object as in the previous step.

```
BehaviorValues.Item("object1") = IxfObject1
```

Save the object.

```
ixfObject.Save
```

Note: Save each object as soon as you are finished creating it.

SmlxfWriter: Creating a ISmlxfXmlAttributeValue

```
Dim XmlAttributeValue as ISmlxfXmlAttributeValue

Set XmlAttributeValue = IxfWriter.DataWriter.Info.CreateXmlAttributeValue

XmlAttributeValue.XML = <p:name>John Bryce<p:name>

XmlAttributeValue.Namespaces.Add 'http://www.vendor.com/ns/personalIdentity',
'p'
```

ISmlxfDataWriter: Writing an Info section

'Optional "Info" section:

```
Set InfoItem = Writer.DataWriter.Info.GetInfoItem("From",
"http://smarteam.com/dev/ixf/test")

InfoItem.ValueType = dtString

InfoItem.Value = "Ann Barkley"

Set InfoItem = Writer.DataWriter.Info.GetInfoItem("To",
"http://smarteam.com/dev/ixf/test")

InfoItem.ValueType = dtString

InfoItem.Value = "Bruce Mayer"

Set InfoItem = Writer.DataWriter.Info.GetInfoItem("Subject",
"http://smarteam.com/dev/ixf/test")

InfoItem.ValueType = dtString

InfoItem.Value = "iXF Example"

DataWriter.Info.Save
```

Note: If you are writing Info to the data file, it must be saved to the data file before saving any object to the data file.

SmlxfReader

An **SmlxfReader** object is used for:

- unpacking an IXF Archive file
- Reading a Schema Document (schema file)
- Reading an IXF Instance Document (data file)
- Optionally, the SmlxfReader can refer to an external schema file.

Object Diagram

The object diagram of SmlxfReader is shown below:

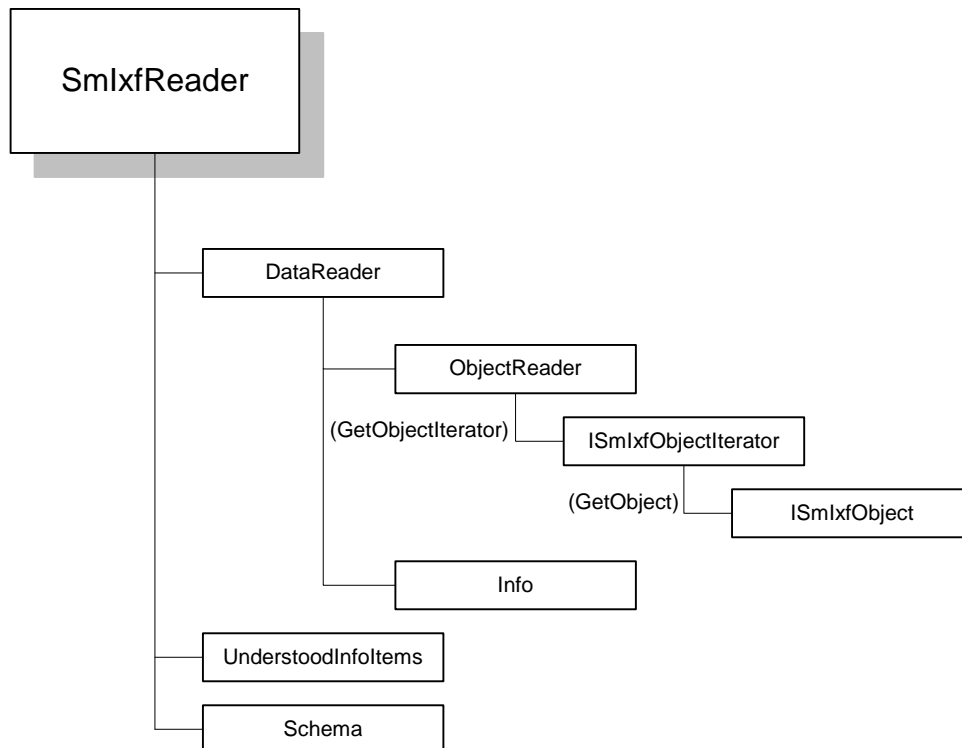


Figure 11-6 SmlxfReader Object Diagram

Properties

The SmIXfReader has the properties

Property	Description
DataReader	Reference to ISmIxfDataReader, which reads the data file. The data file contains a set of objects and miscellaneous data that conform to the data model described in the schema. [Associated files come from the DataReader helper].
UnderstoodInfoltems	Collection of Infoltems that the DataReader declares as understood. Used to validate read-in Infoltems marked as "mustUnderstand".
Schema	Reference to ISmIxfSchema, which holds the definition of the data structure.
ValidateMustUnderstand	If true, validate read-in Infoltems marked as "mustUnderstand" against the UnderstoodInfoltems collection.
InitializationData	Provides access to methods for initializing data for IXF application. Returns ISmIxfInitializationData

Methods

The SmIXfReader has the methods

Methods	Description
OpenIxfArchiveFile	Opens the specified iXF Archive file for reading.
Close	Closes the iXF Archive file for reading.

ISmIxfDataReader

The ISmIxfDataReader object reads the object and miscellaneous data from the data file corresponding to the schema file. The ISmIxfDataReader object includes the ObjectReader property, which is used to read objects from the data file by iteration, using the ObjectsIterator property.

Properties

The ISmIxfDataReader Object has the two properties:

Property	Description
ObjectReader	Returns an ISmIxfObjectReader object
Info	Miscellaneous (Info) information read from the data file.

Obtaining the ISmIxfDataReader Object

To obtain the ISmIxfDataReader Object from the ixfReader Object:

```
Dim DataReader As ISmIxfDataReader

Set DataReader = IxfReader.DataReader
```

ISmIxfObjectReader

The ISmIxfObjectReader object reads objects from the data file. It has one method GetObjectIterator, which returns the object ISmIxfObjectIterator.

Obtaining the ISmIxfObjectReader Object

To obtain the ISmIxfObjectReader Object:

```
Dim IxfObjectReader as ISmIxfReader

Set IxfObjectReader = DataReader.ObjectReader
```

ISmIxfObjectIterator

The ISmIxfObjectIterator reads the objects one-by-one from the data file.

Use the GetObjectIterator method to get an ISmIxfObjectIterator from the ObjectReader as follows:

```
Set ObjectIterator = IxfReader.DataReader.ObjectReader.GetObjectIterator
```

The ISmIxfObjectIterator has one property and three functions:

Property	Description
AtEnd	Indicates whether the iterator has reached the end of the collection.
Method	Description
GetObject	Returns the object to which the iterator is currently pointing.
Next	Sets the iterator to read the next object in the collection

ISmIxfObject

The object represents an individual object read from the data file. See ISmIxfObject under ISmIxfObjectWriter.

Use the GetObject method to get an ISmIxfObject from the ObjectIterator as follows:

```
IxfObject = ObjectIterator.GetObject
```

For an example of how to use the ObjectIterator to read objects, see Common Tasks, SmIxfReader: Reading an iXF Package.

ISmIxfInfo

The ISmIxfInfo object represents miscellaneous information read from the data file. See ISmIxfInfo on page 373.

Note: The ISmIxfInfo information in the iXF Instance file, if it exists, must be read before all object information.

For an example of how to read ISmIxfInfo objects, see Common Tasks, SmIxfReader: Reading an iXF Package.

ISmIxfUnderstoodInfoItems

ISmIxfUnderstoodInfoItems is a collection object, prepared by the user of the ISmIxfReader object, of items of type ISmIxfUnderstoodInfoItem, which denote InfoItems that are required to be understood.

The ISmIxfUnderstoodInfoItems corresponds to a list that specifies those InfoItems that he declares he understands. When the InfoItems are read by the ISmIxfReader, the MustUnderstand property of each InfoItem is matched with the corresponding InfoItem entry in the ISmIxfUnderstoodInfoItems list. If the MustUnderstand property of an InfoItem is true and the corresponding InfoItem entry is not found in ISmIxfUnderstoodInfoItems, the reading process is stopped.

Properties

The ISmIxfUnderstoodInfoItem Object has the two properties:

Property	Description
Name	Name of the understood item.
Namespace	Namespace of the understood item.

ISmIxfSchema

The SmIxfSchema object represents the schema file in the package being read. See ISmIxfSchema on page 356.

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to a SmIxfReader.

SmIxfReader: Reading an iXF Package

As described in the IXF Specification, an iXF Archive file is a zip file containing a data file and possibly a schema file. In order to read an iXF archive file proceed as follows.

Create an IxfReader Object

```
Dim IxfReader as ISmIxfReader
```

```
Set IxfReader = CreateObject("SmartIxf1.SmIxfReader")
```

Open an Ixf archive file:

```
IxfReader.OpenIxfArchiveFile "test.ixf", True
```

Read Info (if it exists)

```
Dim Info as ISmIxfInfo
```

```
Dim SenderName, ReceiverName, Subject as Variant
```

```
Set Info = IxfReader.DataReader.Info
```

```
Set InfoItem = Info.GetInfoItem("From", "http://smarteam.com/dev/ixf/test")

SenderName = InfoItem.Value

Set InfoItem = Info.GetInfoItem("To", "http://smarteam.com/dev/ixf/test")

ReceiverName = InfoItem.Value

Set InfoItem = Info.GetInfoItem("Subject", "http://smarteam.com/dev/ixf/test")

Subject = InfoItem.Value
```

Read Objects:

```
Dim ObjectIterator as ISmIxfObjectIterator
Dim IxfObject as ISmIxfObject

Set ObjectIterator = IxfReader.DataReadet.ObjectReader.GetObjectIterator

While ObjectIterator.AtEnd = False

    Set IxfObject = ObjectIterator.GetObject

    ....

    ObjectIterator.Next

Wend
```

Close the reader object:

```
IxfReader.Close
```

Reading and Writing an External Schema

The SmartIXf library provides two objects for reading and writing an external schema.

SmIxfExternalSchemaWriter

The SmIxfExternalSchemaWriter has the two properties:

Property	Description
Schema	Returns object ISmIxfSchema containing the external schema information
SchemaURI	URI of external schema file.
InitializationData	Provides access to methods for initializing data for IXF applications. Returns ISmIxfInitializationData

The SmIxfExternalSchemaWriter has one method, Save (FileName), which saves the schema to the file FileName.

See ISmIxfSchema on page [356](#), for more information.

SmIxfExternalSchemaReader

The SmIxfExternalSchemaReader handles reading an external iXF schema document

The SmIxfExternalSchemaReader has one method:

- Load(SchemaLocation), which loads the external schema with the specified SchemaLocation into the ISmIxfSchema object.

And one property:

- InitializationData, which provides access to methods for initializing data for IXF applications. Returns ISmIxfInitializationData

See ISmIxfSchema on page [356](#), for more information.

ISmIxfStdHelper

IXF Standard Behaviors, as defined in the IXF Specification, Section 4, are a set of Class Behaviors and Domain Behaviors, which provide common functionality required by many IXF-enabled applications.

The ISmIxfStdHelper object provides methods to simplify and facilitate the usage of IXF Standard Behaviors in the following main functional areas:

- **ISmIxfSchemaHelper** – Adding Standard Behavior definitions to a schema
- **ISmIxfWriterHelper** – Using Standard Behaviors while writing IXF documents
- **ISmIxfReaderHelper** – Using Standard Behaviors while reading IXF documents

Methods

The ISmIxfStdHelper object provides the following methods:

Method	Description
CreateReaderHelper	Creates a reader Helper for using Standard Behaviors while reading IXF documents:
CreateSchemaHelper	Creates a schema Helper for adding Standard Behavior definitions to a schema.
CreateWriterHelper	Creates a writer Helper for using Standard Behaviors while writing IXF documents.xxx
InitializationData	Provides access to methods for initializing data for IXF applications. Returns ISmIxfInitializationData.

Obtaining the SmlxfStdHelper Object

```
Dim StdHelper as ISmIxfStdHelper
```

```
StdHelper = CreateObject("SmartIXF1.SmIxfStdHelper")
```

Standard Behaviors

The SmartIxf Library supports the following mechanisms, which are defined in the iXF standard:

- **Time Stamping** - provides the ability to time-stamp IXF Objects.
- **Change Tracking** - provides a standard mechanism for tracking changes in an IXF Instance Document

File Association -- provides a standard mechanism for:

- Storing file information in the IXF Instance file
- Embedding files in an IXF Archive file
- Associating IXF Objects with files.

Versioning - provides the ability to tag iXF Objects with versioning information.

Links - formalizes and classifies the relationships between objects in an IXF Instance Document.

ISmIxfSchemaHelper

The ISmIxfSchemaHelper object provides methods to support defining Standard Behaviors in a schema.

The ISmIxfSchemaHelper object uses the following naming convention to describe its methods:

Method	Description
Add[std-behavior-name]Support	Adds the classes, class behaviors, and domain behaviors required to support the Standard Behavior in the schema. Note that all implementation details regarding the Classes, which are added by this method to the schema, may change in subsequent versions of this API.
Enable[std-behavior-name]ForClass	Enables standard behavior functionality in a specific user-defined class.
Is[std-behavior-name]Enabled	Tests a user-defined class to see if it is associated with a Standard Behavior.

Change-Tracking Standard Behavior

The SmartIxf Library provides an implementation of the Change-Tracking Standard Behavior to provide a standard mechanism for tracking changes on objects in an IXF Instance Document, including object creation, object deletion, and attribute value modification.

Methods

The ISmIxfSchemaHelper object provides the following methods to support defining the Change Tracking Standard Behavior in a schema:

Method	Description
AddDefaultChangesSupport	Adds the Change-Tracking Standard Behavior to the schema as shown in Change-Tracking Standard Behavior.
EnableChangeTrackingForClass	Add the TrackChanges Class Behavior to the specified user-defined class, enabling the class to be change-tracked.
IsChangeTrackingEnabled	Returns true if the specified user-defined class supports Change Tracking Standard Behavior.

Behaviors

The following table shows the domain and class behaviors added to the schema that supports the implementation of the Change Tracking Standard Behavior:

Domain Behavior	
Name	URI
Change Tracking	ixfstdns:/domainBehaviors/changeTracking/1.0

Class Behaviors	
Name	URI
change	ixfstdns-c:/changeTracking/1.0#change
objectDeleted	ixfstdns-c:/changeTracking/1.0#objectDeleted
objectValue Modified	ixfstdns-c/ /changeTracking/1.0#objectValueModifi
objectCreated	ixfstdns-c/ /changeTracking/1.0#objectCreated
transaction	ixfstdns-c/ /changeTracking/1.0#transaction

Note: An object can be change-tracked only if it instantiates a class that is enabled for change-tracking.

File Association Standard Behavior

The SmartIxf Library provides an implementation of the File Association Standard Behavior to provide a standard mechanism for:

- Storing file information in an IXF Instance file, including
 - File Name
 - Physical Location
 - MIME Content Type
 - Associating an IXF Object with a specific file
 - Distinguishing between main and secondary files
 - Embedding files in an IXF Archive file

Methods

The ISmIxfSchemaHelper object provides the following methods to support defining the File Association Standard Behavior in a schema:

Method	Description
AddDefaultFilesSupport	Adds the Files Standard Behavior to the schema as shown in Error! Not a valid result for table. , including the C Behaviors:
EnableFileAssociationForClass	Add the File Association Class Behavior to the specified user-defined class.
IsFileAssociationEnabled	Returns true if the specified user-defined class supports the Files Standard Behavior.

Behaviors

The following table shows the domain and class behaviors added to the schema that supports the implementation of the File Association Standard Behavior:

Domain Behavior	
Name	URI
Files	<ixfstdns-d>/domainBehaviors/files/1

Class Behaviors	
Name	URI

File Association	<ixfstdns-c>/files/1.0#fileAssociatio
File Description	<ixfstdns-c>/files/1.0#fileDescriptio
Main File	ixfstdns-c>/files/1.0#mainFile
Secondary File	<ixfstdns-c>/files/1.0#secondaryFile
Transaction	<ixfstdns-c>/changeTracking/1.0#transaction

Note: An object can be associated with a file only if it instantiates a class that is enabled for File Association.

Versioning Standard Behavior

The SmartIxf Library provides an implementation of the Versioning Standard Behavior to provide the ability to tag IXF Objects with versioning information, enabling you to identify successive revisions of the same master entity.

The versioning information for an object includes the version identifier of the current version of the object and the version identifier of its previous version.

The Versioning Standard Behavior is different from the Change-Tracking Standard Behavior: it just assigns version numbers without tracking the changes between versions.

Methods

The ISmIxfSchemaHelper object provides the following methods to support defining the Versioning Standard Behavior in a schema:

Method	Description
AddDefaultVersioningSupport	Adds the Versioning Standard Behavior to the schema.
EnableVersioningForClass	Add the Versioning Class Behavior to the specified user-defined class.
IsVersioningEnabled	Returns true if the specified user-defined class supports the Versioning Standard Behavior.

Behaviors

The following table shows the class behaviors added to the schema that support the Versioning Standard Behavior:

Class Behaviors	
Name	URI
Versioning	<ixfstdns-c>/versioning/1.0#versioning

Note: An object can be versioned only if it instantiates a class that is enabled for Versioning.

TimeStamp Standard Behavior

The SmartIxf Library provides an implementation of the TimeStamp Standard Behavior to provide the ability to tag IXF Objects with TimeStamp information, enabling you to mark the time of object creation.

Methods

The ISmIxfSchemaHelper object provides the following methods to support defining the TimeStamp Standard Behavior in a schema:

Method	Description
AddDefaultTimeStampSupport	Adds the TimeStamp Standard Behavior to the schema, as shown in TimeStamp Standard Behavior.
EnableTimeStampForClass	Add the enabler TimeStamp Class Behavior to the specified user class.
IsTimeStampEnabled	Returns true if the specified user class supports TimeStamp Standard Behavior.

Behaviors

The following table shows the class behaviors added to the schema that supports the TimeStamp Standard Behavior:

Class Behaviors	
Name	URI
Time Stamping	<ixfstdns-c>/timeStamp/1.0#timeStamp

Note: An object can be time stamped only if it instantiates a class that is enabled for TimeStamp.

Obtaining the ISmIxfSchemaHelper Object

To create an ISmIxfSchemaHelper Object:

```
Dim SchemaHelper as ISmIxfSchemaHelper

'Create Schema Helper:

Set SchemaHelper = StdHelper.CreateSchemaHelper(Schema)
```

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to a ISmIxfSchemaHelper.

ISmIxfSchemaHelper:

Add supported standard behaviors to the schema

Write basic schema, see Common Tasks in ISmIxfSchema section.

Get SchemaHelper object

```
Dim StdHelper as ISmIxfStdHelper

Dim SchemaHelper as ISmIxfSchemaHelper

'Create stdHelper:

StdHelper = CreateObject("SmartIXF1.SmIxfStdHelper")

'Create Schema Helper:

Set SchemaHelper = StdHelper.CreateSchemaHelper(Schema)
```

Add support for standard behaviors to the schema:

```
'Add support for Standard Behaviors:

SchemaHelper.AddDefaultChangesSupport

SchemaHelper.AddDefaultFilesSupport

SchemaHelper.AddDefaultVersioningSupport

SchemaHelper.AddDefaultTimeStampSupport
```

Enable a user-defined class to support standard behaviors

`SchemaHelper.EnableFileAssociationForClass (IxfClass)`

`SchemaHelper.EnableChangeTrackingForClass (IxfClass)`

`SchemaHelper.EnableTimeStampForClass (IxfClass)`

ISmIxfWriterHelper

The ISmIxfWriterHelper object supports writing Standard Behaviors attribute information when writing a data file.

Object Diagram

The object diagram of ISmIxfWriterHelper is shown below:

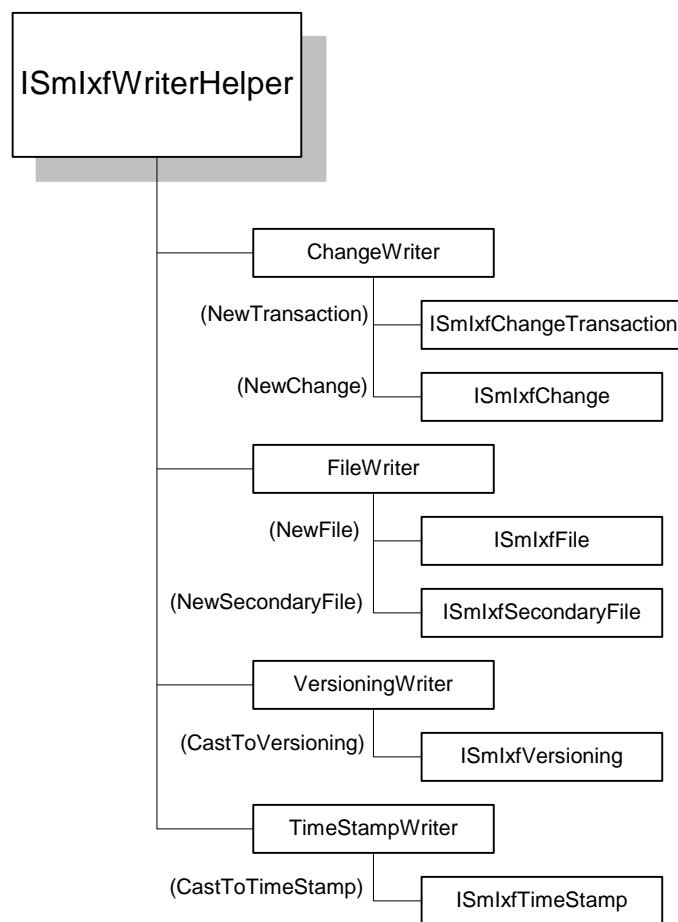


Figure 11-7 ISmIxfWriterHelper Object Diagram

Properties

Four WriterHelper properties are provided corresponding to the supported Standard Behaviors:

Property	Description
ChangeWriter	Provides methods for writing Change-Tracking information
FileWriter	Provides methods for writing File Association information
VersioningWriter	Provides methods for writing Versioning information
TimeStampWriter	Provides methods for writing TimeStamp information

Obtaining the ISmIxfWriterHelper Object

To create an ISmIxfWriterHelper Object:

```
Dim WriterHelper as ISmIxfWriterHelper
```

```
    'Create Writer Helper:
```

```
Set WriterHelper = StdHelper.CreateWriterHelper(IxfWriter)
```

ISmIxfChangeWriter

The ISmIxfChangeWriter writes change-tracking information to the data file.

Note: In order to use the methods of the ISmIxfChangeWriter you need to have added the ChangeTracking Standard Behavior support in the schema, using the SchemaHelper method AddDefaultChangesSupport. In addition, in order to use Change-Tracking on a specific IxfObject, you need to have enabled the ChangeTracking Standard Behavior support for the class that this object instantiates, using the SchemaHelper method EnableChangeTrackingForClass (see Change-Tracking Standard Behavior)

Obtaining the ISmIxfChangeWriter Object

```
Dim ChangeWriter as ISmIxfChangeWriter
```

```
Set ChangeWriter = WriterHelper.ChangeWriter
```

Methods

The ISmIxfChangeWriter object provides the following methods to write Change-Tracking information to the data file:

Method	Description
NewChange	Creates an ISmIxfChange object
NewTransaction	Creates an ISmIxfChangeTransaction

ISmIxfChangeTransaction

The ISmIxfChangeTransaction Object is a container object that represents a group of changes, where each change is represented by a ISmIxfChange object. A ISmIxfChange object is linked to an ISmIxfChangeTransaction Object by its Transaction Id property.

Obtaining a ISmIxfChangeTransaction Object

To create a ISmIxfChangeTransaction Object:

```
Dim Transaction as ISmIxfChangeTransaction
```

```
Set Transaction = ChangeWriter.NewTransaction(Id)
```

Note: ISmIxfChangeTransaction objects can also be obtained through ISmIxfChangeReader Object. See ChangeReader.

Save a ChangeTransaction to the data file as follows:

```
ChangeTransaction.Save
```


Properties

The ISmIxfChangeTransaction Object has the properties

Property	Description
Id	Object Id of the SmlxfChangeTransaction object. Has a valid NCName.
ParentTransactionId	Object Id of parent SmlxfChangeTransaction object in file, if it exists
PreviousTransactionId	Object Id of the previous SmlxfChangeTransaction object in the file, if it exists.

Methods

The ISmIxfChangeTransaction Object has the methods

Property	Description
Save	Saves the ChangeTransaction to the data file.

ISmIxfChange

The ISmIxfChange object represents an individual change on an object, and includes change-tracking information. Each ISmIxfChange object is associated with some ISmIxfChangeTransaction object by the Transaction Id property. Therefore, an ISmIxfChangeTransaction object needs to be created first.

Three types of changes are tracked: object creation, object deletion and object modification. The actual changes for each type are represented by three separate objects, which are returned as properties of ISmIxfChange:

- **ISmIxfObjectCreated** – contains the Object Id of the created object
- **ISmIxfObjectDeleted** – contains a reference to the deleted object
- **ISmIxfObjectValueModified** – contains the Id of the modified object and contains a list of the object attributes that were changed, including their previous values.

Object Diagram

The object diagram of ISmIxfChange is shown below:

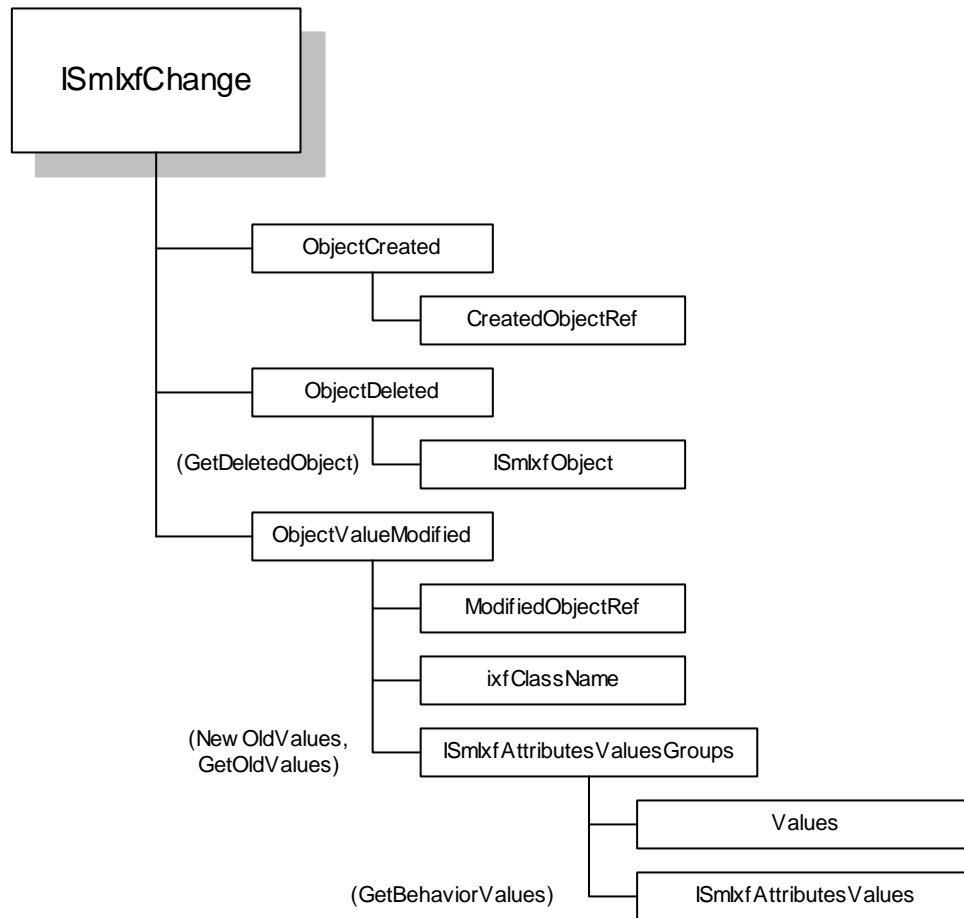


Figure 11-8 ISmIxfChange Object Diagram

Obtaining a ISmIxfChange Object

To create a ISmIxfChange Object:

```
Dim Change as ISmIxfChange
```

```
Set Change = ChangeWriter.NewChange(Id, ChangeType, TransactionId)
```

Note: ISmIxfChange objects can also be obtained through ISmIxfChangeReader Object. See ChangeReader.

Properties

The ISmIxfChange object has the properties:

Property	Description
Id	Object Id of ISmIxfChange object. Has to be a valid NCName.
TransactionId	Object Id of ISmIxfChangeTransaction to which this change belongs
ChangeType	The type of the SmlxfChange, one of ChangeTypeEnum with the following possible values: <ul style="list-style-type: none">- ctObjectCreated- ctObjectDeleted- ctObjectValueModified
ObjectCreated	Returns ISmIxfObjectCreated object (see below.) Can be accessed only when the Change is of type ctObjectCreated
ObjectDeleted	Returns ISmIxfObjectDeleted object (see below.) Can be accessed only when the Change is of type ctObjectDeleted
ObjectValueModified	Returns ISmIxfObjectValueModified object (see below.) Can be accessed only when the Change is of type ctObjectValueModified.
PreviousChangeId	The Object Id of the previous SmlxfChange in time.
PreviousChangeIdPerObject	The Object Id of the previous SmlxfChange on the same object.
IxfObject	Pointer to the ISmIxfObject that is wrapped by the current ISmIxfChange object

Methods

The ISmIxfChange Object has the methods

Property	Description
Save	Saves the Change to the data file.

ISmIxfObjectCreated

The ISmIxfObjectCreated represents a change of type ctObjectCreated. This change is meant to point to a new object that was created and added to an already existing set of objects.

Obtaining a ISmIxfObjectCreated Object

ISmIxfObjectCreated object is obtained through ISmIxfChange object of type ctObjectCreated:

```
Dim Change as ISmIxfChange

Dim ObjectCreated as ISmIxfObjectCreated

Set Change = ChangeWriter.NewChange(Id, ctObjectCreated, TransactionId)

Set ObjectCreated = Change.ObjectCreated
```

The ISmIxfObjectCreated object has the properties:

Property	Description
CreatedObjectRef	The Id of the created object

Example

```
ObjectCreated.CreatedObjectRef = "1"
```

ISmIxfObjectDeleted

The ISmIxfObjectDeleted represents a change of type ctObjectDeleted. This change is meant to hold the information of an object that was deleted from the data objects set.

Obtaining a ISmIxfObjectDeleted Object

ISmIxfObjectDeleted object is obtained through ISmIxfChange object of type ctObjectDeleted:

```
Dim Change as ISmIxfChange

Dim ObjectDeleted as ISmIxfObjectDeleted

Set Change = ChangeWriter.NewChange(Id, ctObjectDeleted, TransactionId)

Set ObjectDeleted = Change.ObjectDeleted
```

Methods

The ISmIxfObjectDeleted object has the methods:

Method	Description
SetDeletedObject	Sets the deleted object as the object referenced by the SmlxfChange
GetDeletedObject	Returns an ISmIxfObject, which is the deleted object.

Example

```
ObjectDeleted.SetDeletedObject (IxfObject)
```

ISmIxfObjectValueModified

The ISmIxfObjectValueModified represents a change of type ctObjectValueModified; it contains the previous values of object attributes that were modified, including both class attributes and behavior attributes.

You do not load individual previous object attribute values directly into the ISmIxfObjectValueModified change object. Instead, you load the previous object attribute values, for the attributes that changed, into the intermediate object ISmIxfAttributesValuesGroups and then map this object to the ISmIxfObjectValueModified object using the method SetOldValues, as described below.

Similarly, when you want to access the previous object attribute values in a ISmIxfObjectValueModified change object, you use the GetOldValues method to extract the information into a ISmIxfAttributesValuesGroups object.

An empty intermediate ISmIxfAttributesValuesGroups object can be created from the ISmIxfObjectValueModified object using the method NewOldValues.

Obtaining a ISmIxfObjectValueModified Object

ISmIxfObjectValueModified object is obtained through ISmIxfChange object of type ctObjectValueModified:

```
Dim Change as ISmIxfChange  
  
Dim ObjectDeleted as ISmIxfObjectValueModified  
  
Set Change = ChangeWriter.NewChange(Id, ctObjectValueModified, TransactionId)  
  
Set ObjectValueModified = Change.ObjectValueModified
```

Properties and Methods

The ISmIxfObjectValueModified object has the following properties and methods:

Property	Description
ModifiedObjectRef	The Object Id of the modified object
IxfClassName	The name of the class that the modified object instantiates
Method	Description
NewOldValues	Creates and returns an ISmIxfAttributesValuesGroups object – empty collection of attribute values to be filled by the user with values of the modified object prior to the change (old values)
SetOldValues	Sets the old values of the SmlxfObjectValueModified object to the values specified in the OldValues argument collection, when OldValues was filled in by the user.
GetOldValues	Returns the collection of attribute values of the modified object prior to the change represented by SmlxfObjectValueModified

ISmIxfAttributesValuesGroups

Collection of ISmIxfAttributesValues objects, where each ISmIxfAttributesValues object is a group of either class attributes or behavior attributes.

Properties and Methods

The ISmIxfAttributesValuesGroups object has the following properties and methods:

Property	Description
Values	An SmlxfAttributesValues object that represents class attribute values (see ISmIxfAttributesValues.)
Method	Description
GetBehaviorValues	An SmlxfAttributesValues object that represents ClassBehavior attributes values (see ISmIxfAttributesValues.)

Example

```
Dim OldValues as ISmIxfAttributesValuesGroups
```

```
ObjectValueModified.ModifiedObjectRef = "1"
```

```
ObjectValueModified.IxfClassName = "DocumentMaster"
```

```
Set OldValues = ObjectValueModified.NewOldValues

'Inserting an old value for a class attribute that was modified:

OldValues.Values.Item("DocumentName") = "MyDocument"

ObjectValueModified.SetOldValues(OldValues)
```

When you finish creating the change, you need to call the Save method of the ISmIxfChange object for all the change details to be saved to the data file:

```
Change.Save
```

For an example of writing a change, see Common Tasks, ISmIxfChangesWriter: Writing a change.

ISmIxfFileWriter

The FileWriter lets you include a file as part of the IXF data. In order to include a file, you must create a ISmIxfFile object to represent it. The ISmIxfFile object contains detailed information about the file, such as its name and location.

The physical file, which is represented by the ISmIxfFile object, can be embedded into the iXF Archive file, using the EmbedFile method.

In addition, you can associate the file with an existing IxfObject such as a Document.

Note: In order to use the methods of the ISmIxfFileWriter you need to have added the File Association Standard Behavior support in the schema. Specifically you should include the AddDefaultFilesSupport method in the schema (see

File Association Standard Behavior.) If you want to use the FileAssociation capability you need to include additional methods, as described below.

Object Diagram

The object diagram of ISmIxfFileWriter is shown below:

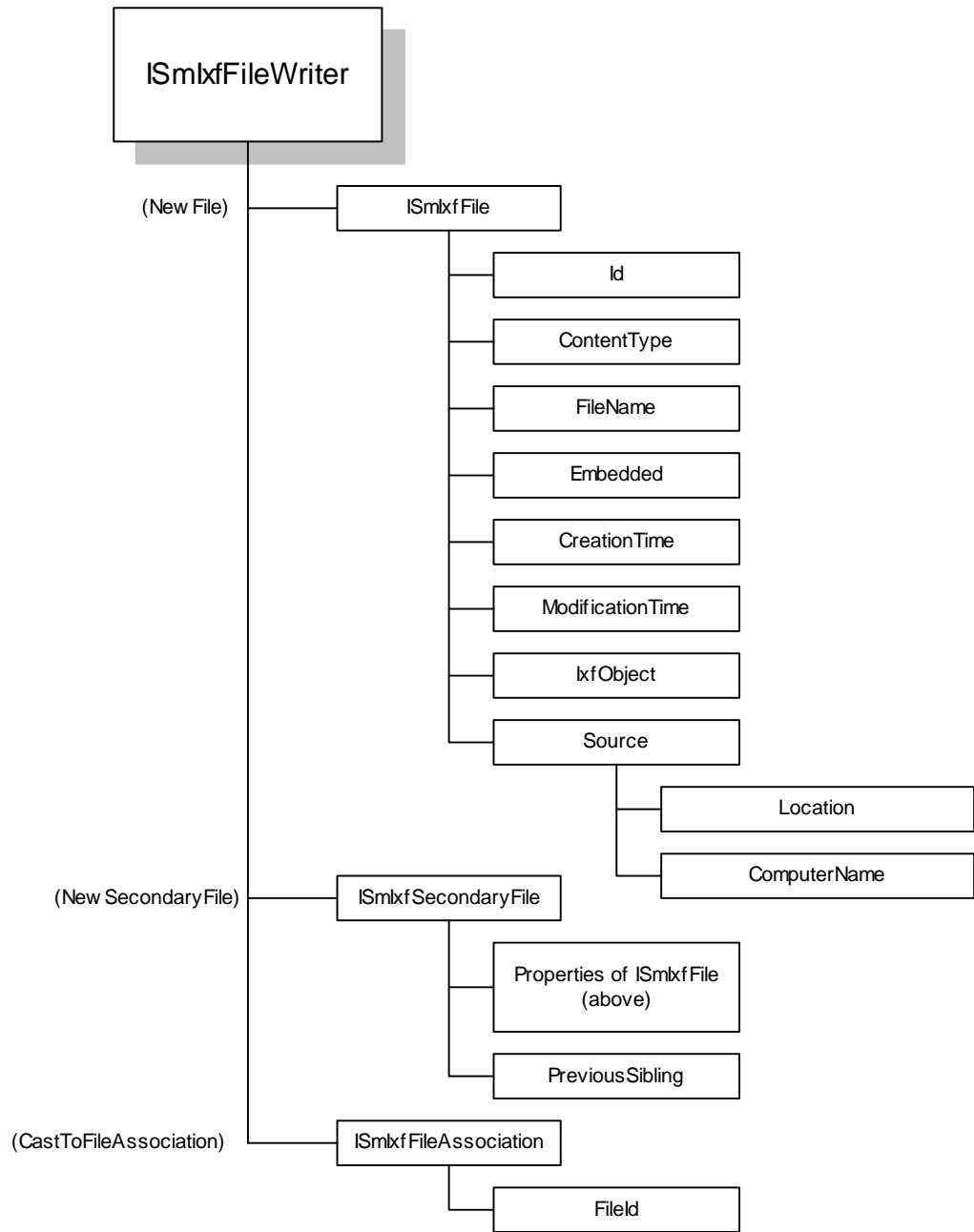


Figure 11-9 *ISmIxfFileWriter* Object Diagram

Obtaining the ISmIxfFileWriter Object

Create a ISmIxfFileWriter object as follows:

```
Dim FileWriter as ISmIxfFileWriter  
Set FileWriter = WriterHelper.FileWriter
```

Methods

The ISmIxfFileWriter object provides methods to write file information to the data file:

Method	Description
NewFile	Creates a new ISmIxfFile object
NewSecondaryFile	Creates a new ISmIxfSecondaryFile object
EmbedFile	Embeds a file in an IXF Archive file.
EmbedSecondaryFile	Embeds an associated file in an IXF Archive file.
CastToFile	Converts a SmlxfObject to a SmlxfFile object. The SmlxfObject to be associated with a file must support File Association Standard Behavior, otherwise the method returns null.
CastToSecondaryFile	Converts a SmlxfObject to a SmlxfSecondaryFile object. The SmlxfObject to be associated with a file must support File Association Standard Behavior, otherwise the method returns null.
CastToFileAssociation	Converts a SmlxfObject to a SmlxfFileAssociation object, which is used to associate the object with a file. The SmlxfObject to be associated with a file must support File Association Standard Behavior, otherwise the method returns null.

Note: In order to use the methods CastTo... you need to have enabled the File Association Standard Behavior support for the class that the IxfObject instantiates, by using the method EnableFileAssociationForClass (see

File Association Standard Behavior.)

ISmIxfFile

An ISmIxfFile object represents a primary IXF file and contains its information (see also ISmIxfSecondaryFile).

Obtaining a ISmIxfFile Object

To create a new ISmIxfFile object:

```
Dim File as ISmIxfFile
```

```
Set File = FileWriter.NewFile(Id)
```

Note: ISmIxfFile object can also be obtained through ISmIxfFileReader Object.
See FileReader.

Save the file object to the data file after creating it, optionally embedding the physical file into the IXF Archive:

```
FileWriter.EmbedFile(File)
```

```
File.Save
```

Where, if used, the EmbedFile method needs to be called prior to the Save method. The File parameter is an existing ISmIxfFile object.

Note: The Save action itself does not embed the physical file to the iXF Archive file.

For an example of how to write and embed a file, see [Common Tasks](#), ISmIxfFileWriter: Writing and embedding a file

Properties

The ISmIxfFile object has the properties:

Property	Description
Id	Input string that uniquely identifies the file object within the IXF Instance Document. Must be a valid NCName.
FileName	Specifies the name of the file
ContentType	The file MIME content type
CreationTime	The file creation time (TDateTime)
ModificationTime	The file last modification time (TDateTime)
Embedded	Indicates whether or not the file is embedded in the iXF archive
IxfObject	Pointer to the ISmIxfObject that is wrapped by the current ISmIxf object
Source	The physical location of the file (ISmIxfSource)

ISmIxfSecondaryFile

The ISmIxfSecondaryFile object is provided to handle sequences of files. It represents any member of a sequence of files that is not the first file. The position of a ISmIxfSecondaryFile object in the sequence is determined by its "PreviousSibling" property, which is the Id of the previous file in the sequence.

The ISmIxfSecondaryFile object has the same set of properties as the ISmIxfFile object, shown above, except for the addition of the "PreviousSibling" property.

For example, a sequence of three files might be used to contain the information from one scanned picture.

```
File1 (ISmIxfFile) - PreviousSibling = ""
```

```
File2 (ISmIxfSecondaryFile) - PreviousSibling = "File1"
```

```
File3 (ISmIxfSecondaryFile) - PreviousSibling = "File2"
```

The ISmIxfSecondaryFile object represents the information about a secondary file that is written to the data file.

Creating a New Secondary File:

```
Dim SecondaryFile as ISmIxfSecondaryFile
```

```
Set SecondaryFile = FileWriter.NewSecondaryFile(Id, PreviousSibling)
```

Note: ISmIxfSecondaryFile object can also be obtained through ISmIxfFileReader Object. See FileReader.

The file object has to be saved to the data file after finished creating it:

```
FileWriter.EmbedSecondaryFile(SecondaryFile)

SecondaryFile.Save
```

where you need to call the `EmbedSecondaryFile` method prior to calling the `Save` method; the file parameter is an existing `ISmIxfSecondaryFile` object.

Note: The save action does not embed the physical file to the iXF Archive file.

ISmIxfFileAssociation

The `ISmIxfFileAssociation` object represents an iXF File Association class behavior, which supports associating an iXF Object, such as a Document with a specific file.

The association between file and object is established between a `FileAssociation` object, which is created from the `IxfObject`, and the `ISmIxfFile` object that represents the file.

The object that is to be associated with a file is cast into an object of type `ISmIxfFileAssociation` using the `FileWriter` method `CastToFileAssociation`. The link from object to file is provided through the `FileId` property of the `ISmIxfFileAssociation` object, which is set to the `Id` of the `ISmIxfFile` object.

Obtaining a FileAssociation Object

```
Dim FileAssociation as ISmIxfFileAssociation

Set FileAssociation = FileWriter.CastToFileAssociation(IxfObject)
```

Properties

The `ISmIxfFileAssociation` object has one property: the `Id` of the associated file object, which provides the association between object and file.

For an example of writing a file, see [Common Tasks](#), `ISmIxfFileWriter`: Writing and embedding a file

ISmIxfVersioningWriter

The ISmIxfVersioningWriter provides the ability to add versioning information to an ISmIxfObject. The ISmIxfVersioningWriter is useful when you need to identify successive revisions of the same entity, for example, successive versions of the same document.

The versioning information itself is represented by an ISmIxfVersioning object, which is obtained from the ISmIxfObject for which versioning is desired by the VersioningWriter method CastToVersioning. The ISmIxfVersioning object includes the version identifiers of the current version and previous version of the entity.

The ISmIxfVersioningWriter has one method: CastToVersioning, which converts an ISmIxfObject to ISmIxfVersioning

Note: In order to use the methods of the ISmIxfVersioningWriter you need to have added the Versioning Standard Behavior support in the schema, using the SchemaHelper method AddDefaultVersioningSupport (see [Versioning Standard Behavior](#))

Obtaining the ISmIxfVersioningWriter Object

```
Dim VersioningWriter as ISmIxfVersioningWriter  
Set VersioningWriter = WriterHelper.VersioningWriter
```

ISmIxfVersioning

The ISmIxfVersioning object represents the versioning information for the IxfObject from which it was cast.

Obtaining a Versioning object

```
Dim Versioning as ISmIxfVersioning  
Set Versioning = VersioningWriter.CastToVersioning(IxfObject)
```

Note: In order to use the method CastToVersioning on an IxfObject, you need to have enabled the Versioning Standard Behavior support for the class that IxfObject instantiates, using the SchemaHelper method EnableVersioningForClass (see [Versioning Standard Behavior](#))

Properties

The ISmIxfVersioning object has two properties:

Property	Description
PreviousVersion	The previous version identifier of the IxfObject from which the SmlxfVersioning object was cast.
Version	The current version identifier of the IxfObject from which the SmlxfVersioning object was cast.

For an example of writing a change, see [Common Tasks](#),
ISmIxfVersioningWriter: Versioning an object

ISmIxfTimeStampWriter

The ISmIxfTimeStampWriter provides the ability to add TimeStamp information to an IxfObject, enabling you to mark the time of object creation and modification.

The TimeStamp information itself is represented by an ISmIxfTimeStamp object, which is obtained from the ISmIxfObject for which time-stamping is desired by the TimeStampWriter method CastToTimeStamp. The ISmIxfTimeStamp object includes the creation time and modification time of the ISmIxfObject.

Note: In order to use the methods of the ISmIxfTimeStampWriter you need to have added the TimeStamp Standard Behavior support in the schema, using the SchemaHelper method AddDefaultTimeStampSupport (see [TimeStampReader](#))

Methods

The ISmIxfTimeStampWriter object has one method: CastToTimeStamp, which converts an ISmIxfObject to ISmIxfTimeStamp

Obtaining the ISmIxfTimeStampWriter Object

```
Dim TimeStampWriter as ISmIxfTimeStampWriter  
Set TimeStampWriter = WriterHelper.TimeStampWriter
```

ISmIxfTimeStamp

The ISmIxfTimeStamp object represents the TimeStamp information for the IxfObject from which it was cast.

Obtaining a ISmIxfTimeStamp Object

```
Dim TimeStamp as ISmIxfTimeStamp
```

```
Set TimeStamp = TimeStampWriter.CastToTimeStamp(IxfObject)
```

Note: In order to use the method CastToTimeStamp you need to have enabled the TimeStamp Standard Behavior support for the class that IxfObject instantiates, using the SchemaHelper method EnableTimeStampForClass (see [TimeStamp Standard Behavior](#))

Properties

The ISmIxfTimeStamp object has two properties:

Property	Description
CreationTime	Time of creation of IxfObject.
ModificationTime	Time of modification of IxfObject.

For an example of writing a change, see [Common Tasks](#),
ISmIxfTimeStampWriter: Time-stamping an object

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to the Standard Behavior writers.

ISmIxfChangesWriter: Writing a change

```
Dim StdHelper as ISmIxfStdHelper

Dim WriterHelper as ISmIxfWriterHelper

Dim ChangeWriter as ISmIxfChangeWriter

Dim DocumentObject as Object


'Create stdHelper:

StdHelper = CreateObject("SmartIXF1.SmIxfStdHelper")


'Create Writer Helper:

Set WriterHelper = StdHelper.CreateWriterHelper(IxfWriter)


'Create Change Writer:

Set ChangeWriter = WriterHelper.ChangeWriter


'Create ChangeTransaction:

Set Transaction = ChangeWriter.NewTransaction("1")

Transaction.Save


'Create a new object:

Set IxfObject = IxfWriter.DataWriter.ObjectWriter.NewObject("Document", "3");

...

IxfObject.Save;
```

'Create a Change for the created object:

Set Change = ChangeWriter.NewChange("2"; ctObjectCreated; "1")

Set Change.ObjectCreated.CreatedObjectRef = "3"

Change.Save

**ISmIxfFileWriter:
Writing and embedding a file**

```
Dim StdHelper as ISmIxfStdHelper

Dim WriterHelper as ISmIxfWriterHelper

Dim IxfWriter as ISmIxfWriter

Dim FileWriter as ISmIxfFileWriter

Dim File as ISmIxfFile


'Create stdHelper:

StdHelper = CreateObject("SmartIXF1.SmIxfStdHelper")


'Create a writer:

IxfWriter = CreateObject("SmartIXF1.SmIxfWriter")


'Create Writer Helper:

Set WriterHelper = StdHelper.CreateWriterHelper(IxfWriter)


'Create File Writer:

Set FileWriter = WriterHelper.FileWriter


'Create File:

Set File = FileWriter.NewFile("1")

File.FileName = "design.doc"

File.SetSource = "c:\MyDocuments\design.doc"

FileWriter.EmbedFile(File)

File.Save
```

ISmIxfFileWriter: Associating an object with a file

```
Dim FileAssociation as ISmIxfFileAssociation

Dim DocumentObject as ISmIxfObject

'Create a user-defined object:

Set DocumentObject =
IxfWriter.DataWriterObjectWriter.newObject("DocumentMaster", "2")

.....

'Cast the user-defined object to a file association object:

FileAssociation = FileWriter.CastToFileAssociation(DocumentObject)

FileAssociation.FileId = "1"
```

ISmIxfVersioningWriter: Versioning an Object

```
Dim StdHelper as ISmIxfStdHelper

Dim WriterHelper as ISmIxfWriterHelper

Dim IxfWriter as ISmIxfWriter

Dim VersioningWriter as ISmIxfVersioningWriter

Dim Versioning as ISmIxfVersioning

Dim DocumentObject as ISmIxfObject

'Create stdHelper:

StdHelper = CreateObject("SmartIXF1.SmIxfStdHelper")

'Create a writer:

IxfWriter = CreateObject("SmartIXF1.SmIxfWriter")
```

```
'Create Writer Helper:

Set WriterHelper = StdHelper.CreateWriterHelper(IxfWriter)

'Create Versioning Writer:

Set VersioningWriter = WriterHelper.VersioningWriter

'Create a user-defined object:

Set DocumentObject =
IxfWriter.DataWriterObjectWriter.NewObject("DocumentMaster", "4")

.....

'Cast the user-defined object to versioning object:

Versioning = VersioningWriter.CastToVersioning(DocumentObject)

Versioning.Version = "2.0"

Versioning.PreviousVersion = "1.0"
```

ISmIxfTimeStampWriter: Time-stamping an object

```
Dim StdHelper as ISmIxfStdHelper

Dim WriterHelper as ISmIxfWriterHelper

Dim TimeStampWriter as ISmIxfTimeStampWriter

'Create stdHelper:

StdHelper = CreateObject("SmartIXF1.SmIxfStdHelper")

'Create Writer Helper:

Set WriterHelper = StdHelper.CreateWriterHelper(IxfWriter)

Set TimeStampWriter = WriterHelper.TimeStampWriter

'Create TimeStamp:

TimeStamp = TimeStampWriter.CastToTimeStamp(DocumentObject)

TimeStamp.CreationTime = now
```

ISmIxfReaderHelper

The ISmIxfReaderHelper object supports reading Standard Behaviors information from a data file.

Identifying and Restoring Read-In Objects

The Standard Behaviors object data is written by the DataWriter to the IXF Instance data file as the original object types such as ISmIxfChange, ISmIxfFile, ISmIxfVersioning, as described in the section on the ISmIxfWriterHelper.

However, the same object data is read by the DataReader from the IXF Instance data file as generic ISmIxfObject objects rather than as the object types that were originally written to the data file. To identify and restore the original object types, the ReaderHelper provides methods for casting the ISmIxfObject objects read from the data file back to the same type of objects that were written.

For each object read in, you need to run all the cast methods on it successively. When a specific cast method returns a non-null result, you have identified and restored the object.

Object Diagram

The object diagram of ISmIxfReaderHelper is shown below:

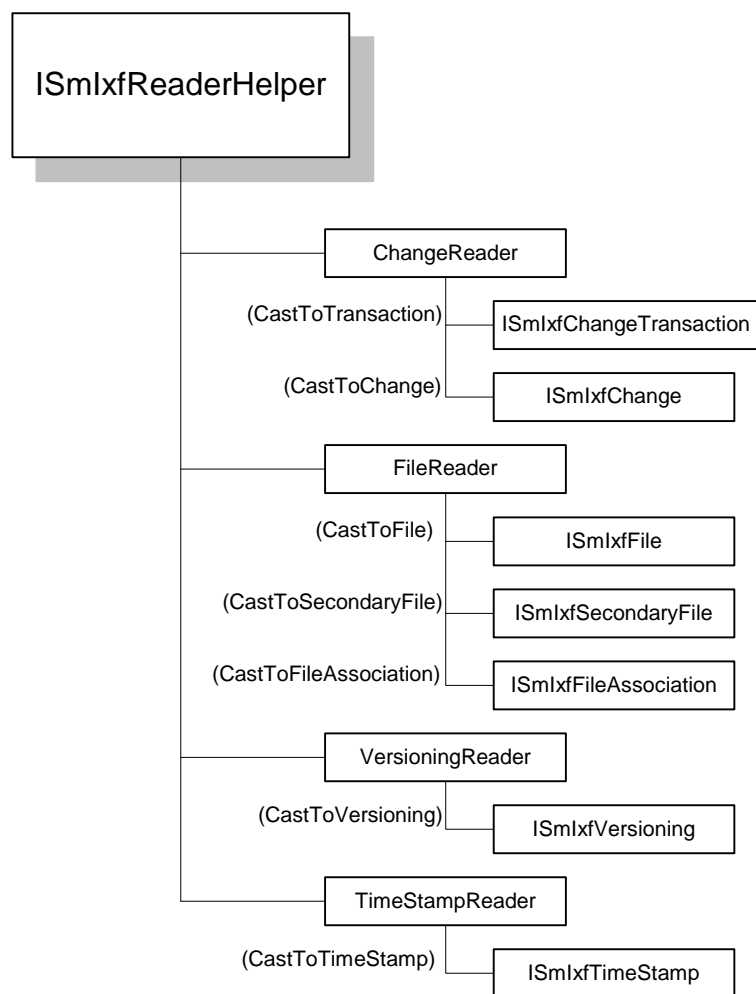


Figure 11-10 ISmIxfReaderHelper Object Diagram

Properties

Four ReaderHelper properties are provided corresponding to the supported Standard Behaviors:

Property	Description
ChangeReader	Provides methods for reading Change-Tracking objects
FileReader	Provides methods for reading File Association objects
VersioningReader	Provides methods for reading Versioning objects
TimeStampReader	Provides methods for reading TimeStamp objects

Obtaining the ISmIxfReaderHelper Object

To obtain an ISmIxfReaderHelper Object:

```
Dim ReaderHelper as ISmIxfReaderHelper  
Set ReaderHelper = StdHelper.CreateReaderHelper(IxfReader)
```

ChangeReader

The ChangeReader helps to read the change-tracking objects ISmIxfChange and ISmIxfChangeTransaction from the Ixf Archive data file. The ChangeReader identifies and restores the original object types, by casting the ISmIxfObject objects read from the Ixf Archive data file back to the same type of objects that were written.

Obtaining the ISmIxfChangeReader Object

```
Dim ChangeReader as ISmIxfChangeReader  
Set ChangeReader = ReaderHelper.ChangeReader
```


Methods

The ISmIxfChangeReader object provides the following methods to read Change-Tracking information from the data file.

Method	Description
CastToChange	Converts an ISmIxfObject to ISmIxfChange
CastToTransaction	Converts an ISmIxfObject to ISmIxfChangeTransaction

Note: In order to use these methods, you need to have added the Change-Tracking Standard Behavior support in the schema, using the SchemaHelper method AddDefaultChangesSupport (see [Change-Tracking Standard Behavior](#))

ISmIxfChangeTransaction

To cast an object to a ChangeTransaction object:

```
Dim Transaction as ISmIxfChangeTransaction  
  
Set Transaction = ChangeReader.CastToTransaction(IxfObject)
```

See ISmIxfChangeTransaction section under ISmIxfChangeWriter for details about the ISmIxfChangeTransaction properties.

ISmIxfChange

To cast an object to a Change object:

```
Dim Transaction as ISmIxfChangeTransaction  
  
Set Transaction = ChangeReader.CastToTransaction(IxfObject)
```

See ISmIxfChange section under [ISmIxfChangeWriter](#) section for details about the ISmIxfChange properties.

For an example of how to use ISmIxfReaderHelper to read Change objects, see [Common Tasks](#), Reading and Casting objects to File and to Change objects

FileReader

The `FileReader` helps to read the File Association objects `ISmIxfFile`, `ISmIxfSecondaryFile` and `ISmIxfFileAssociation` from the Ixf Archive data file. The `FileReader` identifies and restores the original object types, by casting the `ISmIxfObject` objects read from the Ixf Archive data file back to the same type of objects that were written.

Obtaining the `ISmIxfFileReader` Object

To obtain the `ISmIxfFileReader` object:

```
Dim FileReader as ISmIxfFileReader  
  
Set FileReader = ReaderHelper.FileReader
```

Methods

The `ISmIxfFileReader` object provides the following methods for reading file information from the data file:

Method	Description
<code>CastToFile</code>	Converts an <code>ISmIxfObject</code> to <code>ISmIxfFile</code>
<code>CastToSecondaryFile</code>	Converts an <code>ISmIxfObject</code> to <code>ISmIxfSecondaryFile</code>
<code>CastToFileAssociation</code>	Converts an <code>ISmIxfObject</code> to an <code>ISmIxfFileAssociation</code> object

Note: In order to use these methods, you need to have added the File Association Standard Behavior support in the schema by including the `AddDefaultFilesSupport` method in the schema (see

[File Association Standard Behavior](#).) In addition, you need to have enabled the File Association Standard Behavior support for the class that `IxfObject` instantiates, by including the method `EnableFileAssociationForClass` (see

File Association Standard Behavior.)

ISmIxfFile

The ISmIxfFile object represents a primary IXF file and contains the file information.

To cast an object to File object:

```
Dim File as ISmIxfFile  
  
Set File = FileReader.CastToFile(IxfObject)
```

See ISmIxfFile section for details about the ISmIxfFile properties.

In order to extract an embedded file from the iXF Archive file, you can use one the following methods of ISmIxfFile object:

```
File.Extract(RootFolder)  
  
File.ExtractToFile(NewFileName)
```

See the reference guide for more details about those functions.

For an example of how to use ISmIxfReaderHelper to read File objects, see [Common Tasks](#), Reading and Casting objects to File and to Change objects

ISmIxfSecondaryFile

To cast an object to a SecondaryFile object:

```
Dim SecondaryFile as ISmIxfSecondaryFile  
  
Set SecondaryFile = FileReader.CastToSecondaryFile(Id, PreviousSibling)
```

See ISmIxfFile section for details about the ISmIxfSecondaryFile properties.

ISmIxfFileAssociation

To cast an object to a FileAssociation object:

```
Dim FileAssociation as ISmIxfFileAssociation  
  
Set FileAssociation = FileWriter.CastToFileAssociation(IxfObject)
```

See ISmIxfFileAssociation for information about the ISmIxfFileAssociation properties.

VersioningReader

The VersioningReader helps to read ISmIxfVersioning objects from the Ixf Archive data file. The VersioningReader identifies and restores the original object types, by casting the ISmIxfObject objects read from the Ixf Archive data file back to the same type of objects that were written.

Obtaining the ISmIxfVersioningReader Object

To obtain a ISmIxfVersioningReader Object:

```
Dim VersioningReader as ISmIxfVersioningReader  
Set VersioningReader = ReaderHelper.VersioningReader
```

Methods

The ISmIxfVersioningReader has one method: CastToVersioning, which converts an ISmIxfObject to a ISmIxfVersioning object.

Note: In order to use this method, you need to have added the Versioning Standard Behavior support in the schema, using the SchemaHelper method AddDefaultVersioningSupport (see Versioning Standard Behavior). In addition, to use CastToVersioning on an IxfObject, you need to have enabled the Versioning Standard Behavior support for the class that IxfObject instantiates, by including the method EnableVersioningForClass (see

ISmIxfVersioning

See the ISmIxfVersioning section under ISmIxfVersioningWriter for information about this object.

TimeStampReader

The TimeStampReader helps to read ISmIxfTimeStamp objects from the Ixf Archive data file. The TimeStampReader identifies and restores the original object types, by casting the ISmIxfObject objects read from the Ixf Archive data file back to the same type of objects that were written.

Obtaining the ISmIxfTimeStampReader Object

```
Dim TimeStampReader as ISmIxfTimeStampReader  
Set TimeStampWriter = WriterHelper.TimeStampReader
```

Methods

The ISmIxfTimeStampReader object has one method: CastToTimeStamp, which converts an ISmIxfObject to a ISmIxfTimeStamp object.

Note: In order to use this method, you need to have added the TimeStamp Standard Behavior support in the schema, using the SchemaHelper method AddDefaultTimeStampSupport (see [TimeStamp Standard Behavior](#)). In addition, to use CastToTimeStamp on an IxfObject, you need to have enabled the TimeStamp Standard Behavior support for the class that IxfObject instantiates, by including the mISmIxfTimeStampWriter.ethod EnableTimeStampForClass (see [TimeStamp Standard Behavior](#))

ISmIxfTimeStamp

See ISmIxfTimeStamp section under the ISmIxfTimeStampWriter object for information about this object.

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to an ISmIxfReaderHelper.

ISmIxfReaderHelper: Reading and Casting objects to File and to Change objects

```
Dim StdHelper as ISmIxfStdHelper

Dim ReaderHelper as ISmIxfReaderHelper

Dim ObjectIterator As ISmIxfObjectIterator

Dim IxfObject Ss ISmIxfObject

Dim Change As ISmIxfChange

Dim ChangeId, CreatedObjectId As Sting

Dim File As ISmIxfFile

'Create stdHelper:

StdHelper = CreateObject("SmartIXFl.SmIxfStdHelper")

'Create Reader Helper:
```



```
Set ReaderHelper = StdHelper.CreateReaderHelper(IxfReader)

'Create ObjectIterator:
Set ObjectIterator = IxfReader.DataReader.ObjectReader.GetObjectIterator

While ObjectIterator.AtEnd = False
    Set IxfObject = ObjectIterator.GetObject

    'Read Change
    Change = ReaderHelper.ChangeReader.CastToChange(IxfObject)
    If Not (Change Is Nothing) Then
        ChangeId = Change.Id
        If Change.ChangeType = ctObjectCreated Then
            CreatedObjectId = Change.ObjectCreated.CreatedObjectRef
        End If
        ...
    End If

    'Read File
    File = ReaderHelper.FileReader.CastToFile(IxfObject)
    If Not (File Is Nothing) Then
        FileName = File.FileName
        File.Extract("Ixf Sample")
        ...
    End If

    ObjectIterator.Next
Wend
```

An IXF Messaging Application

This section presents a sample iXF application, which demonstrates many of the objects, properties and methods described above.

The sample application includes generating and processing an iXF package for a messaging application, using the basic SmartIxf1.0 API functionality. This example is included in the SDK under Samples/SmartIxf/Vb/Sample1.

A simple messaging format is defined, which includes the basic messaging entities: message, attachment and folder.

Messaging Format

Entity	Attributes
Message	<ul style="list-style-type: none">- From- To- Subject- Body- Importance- Time of sending
Folder	<ul style="list-style-type: none">- Name- Creation time
Attachment	<ul style="list-style-type: none">- File reference
FolderLink	<ul style="list-style-type: none">- Parent folder- Child folder
FolderMessageLink	<ul style="list-style-type: none">- Parent folder- Child message
MessageAttachmentLink	<ul style="list-style-type: none">- Parent message- Child attachment

Class Behaviors

The following table lists the ClassBehaviors.

ClassBehavior	URI	Attributes
Message	<code><exemplens-c></code> /messaging#message	From: String, required To: String, required Subject: String, not required Body: String, not required Importance: Integer, not required default value = 0
Folder	<code><exemplens-c></code> /messaging#folder	Name: String, required
Link	<code><ixfstdns-c></code> /links/1.0#link	Object1, Object2
Directed Link	<code><ixfstdns-c></code> /links/1.0#directedLink	Object1, Object2 Directed from Object1 to Object2
Tree Link	<code><ixfstdns-c></code> /links/1.0#treeLink	Object1, Object2 Object1 is the only parent of Object2
TimeStamp	<code><ixfstdns-c></code> /timeStamp/1.0#timeStamp	creationTime modificationTime
FileAssociation	<code><ixfstdns-c></code> /files/1.0#fileAssociation	file

The following abbreviations are used in the table:

<code><exemplens-c></code>	http://example.com/classBehaviors
<code><ixfstdns-c></code>	http://www.ixfstd.org/std/ns/core/classBehaviors

Domain Behaviors

This section describes the Domain Behavior defined for the messaging application. The URI for the Domain Behavior is:
<http://example.com/domainBehaviors/messaging>.

Domain Behavior Definition

The following table defines the Roles and, for each Role, the Class Behaviors that must be implemented by the class, which is mapped to the role. The timeStamp Standard Behavior is only included in the Message and Folder Roles.

Role	Required Class Behaviors
Message	<exemplens-c>/messaging#message <ixfstdns-c>/timeStamp/1.0#timeStamp
Folder	<exemplens-c>/messaging#folder <ixfstdns-c>/timeStamp/1.0#timeStamp
FolderLink	<ixfstdns-c>/links/1.0#link <ixfstdns-c>/links/1.0#directedLink <ixfstdns-c>/links/1.0#treeLink Informal restriction: must point to folder-behavior objects
Attachment	<ixfstdns-c>/files/1.0#fileAssociation
MessageAttachmentLink	<ixfstdns-c>/links/1.0#link <ixfstdns-c>/links/1.0#directedLink informal restriction: parent = message, child = attachment
FolderMessageLink	<ixfstdns-c>/links/1.0#link <ixfstdns-c>/links/1.0#directedLink informal restriction: parent = folder, child = message

Role-to-Class Mapping

The Role-to-Class mapping for the Domain Behavior is:

Role	Class
Message	Message
Folder	Folder
FolderLink	FolderLink
Attachment	Attachment
AttachmentLink	AttachmentLink
FolderMessageLink	FolderMessageLink

Connectivity of Objects

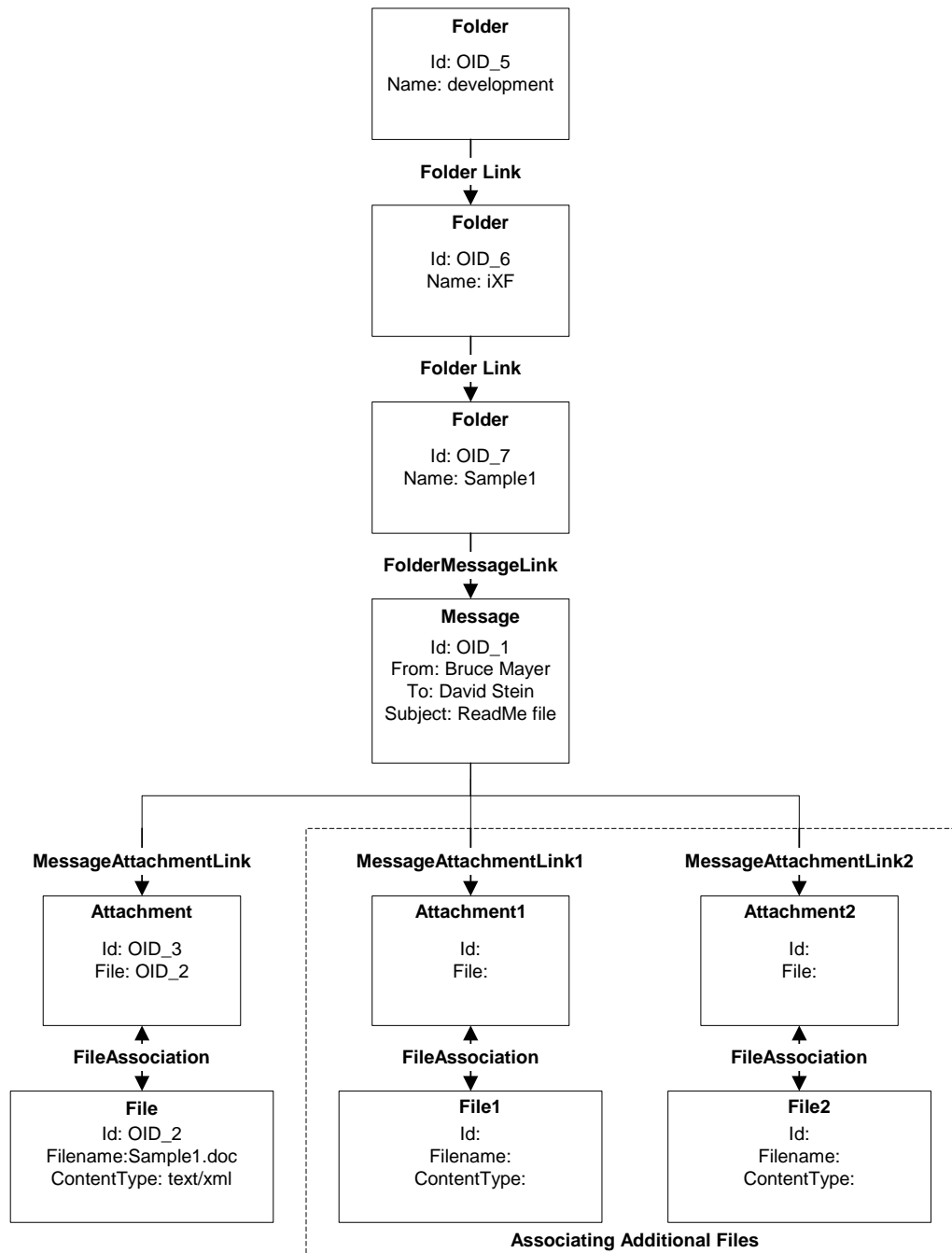
The following diagram shows the connectivity of the basic object and the link objects in the example.

Associating Files with Messages

Note that although the file is associated with the message, the File object is not associated directly to the Message object, but rather through an Attachment object. The File is associated with the Attachment object through the FileAssociation Standard Behavior and the attachment object is linked to the message object with the MessageAttachmentLink.

The reason it is done this way is that the FileAssociation Standard Behavior allows you to associate at most one file with an object enabled for FileAssociation. Thus, to allow for the possibility of associating more than one file to a message, the messaging application has been designed with the intermediate Attachment object and the MessageAttachmentLink object. For each file you want to associate with a message, you create a separate Attachment object and a corresponding MessageAttachmentLink object and follow the procedure of the example.

The figure below shows how you would associate more than one file to the message.



Implementing the Application

This section shows code examples of how the messaging application is implemented. This section does not include all the code in the example, but rather the code needed to illustrate and explain the implementation. For the full code, see the example included in:

SDK/Samples/SmartIxf/Vb/Sample1/Sample1.vbp

Creating the Schema

This section shows how to create the schema for the application and includes the topics:

- [Adding Class Behaviors](#)
- [Adding Classes](#)
- [Adding Domain Behaviors](#)

Adding Class Behaviors

The following functions add the required Class Behaviors to the schema:

Add API-Supported ClassBehaviors

```
Private Sub (Schema As ISmIxfSchema, SchemaHelper As ISmIxfSchemaHelper)
    'Add iXF TimeStamp Standard Class Behavior
    SchemaHelper.AddDefaultTimeStampSupport

    'Add iXF FileAssociation Standard Class Behavior
    SchemaHelper.AddDefaultFilesSupport
End Sub
```

Add Message ClassBehavior

```
Private Sub AddMessageClassBehavior(Schema As ISmIxfSchema)
    Dim IxfClassBehavior As ISmIxfClassBehavior
    Dim IxfAttribute As ISmIxfAttribute
```

```
'Add Message ClassBehavior to Schema ClassesBehaviors  
  
Set IxfClassBehavior = Schema.ClassesBehaviors.Add(mtEmbedded,  
CB_MESSAGE_URI)
```

```
'Add "from" attribute to Message ClassBehavior  
  
Set IxfAttribute = IxfClassBehavior.Attributes.Add("from")  
  
IxfAttribute.TypeDefinition.ValueType = dtString  
  
IxfAttribute.Required = True
```

```
'Add "to" attribute to Message ClassBehavior  
  
Set IxfAttribute = IxfClassBehavior.Attributes.Add("to")  
  
IxfAttribute.TypeDefinition.ValueType = dtString  
  
IxfAttribute.Required = True
```

```
'Add "subject" attribute to Message ClassBehavior  
  
Set IxfAttribute = IxfClassBehavior.Attributes.Add("subject")  
  
IxfAttribute.TypeDefinition.ValueType = dtString  
  
IxfAttribute.Required = False
```

```
'Add "body" attribute to Message ClassBehavior  
  
Set IxfAttribute = IxfClassBehavior.Attributes.Add("body")  
  
IxfAttribute.TypeDefinition.ValueType = dtString  
  
IxfAttribute.Required = False
```

```
'Add "importance" attribute to Message ClassBehavior  
  
Set IxfAttribute = IxfClassBehavior.Attributes.Add("importance")  
  
IxfAttribute.TypeDefinition.ValueType = dtInt
```



```
        IxfAttribute.Required = False

        IxfAttribute.DefaultValue = 0

    End Sub
```

Add Folder ClassBehavior

```
Private Sub AddFolderClassBehavior(Schema As ISmIxfSchema)

    Dim IxfClassBehavior As ISmIxfClassBehavior

    Dim IxfAttribute As ISmIxfAttribute

    'Add Folder ClassBehavior to Schema ClassesBehaviors

    Set IxfClassBehavior = Schema.ClassesBehaviors.Add(mtEmbedded,
    CB_FOLDER_URI)

    'Add "name" attribute to Folder ClassBehavior

    Set IxfAttribute = IxfClassBehavior.Attributes.Add("name")

    IxfAttribute.TypeDefinition.ValueType = dtString

    IxfAttribute.Required = True

End Sub
```

Add Link ClassBehavior

```
Private Sub AddLinkClassBehavior(Schema As ISmIxfSchema)

    Dim IxfClassBehavior As ISmIxfClassBehavior

    Dim IxfAttribute As ISmIxfAttribute

    'Add IXF Standard ClassBehavior "Link" to Schema ClassesBehaviors

    Set IxfClassBehavior = Schema.ClassesBehaviors.Add(mtEmbedded,
    CB_LINK_URI)

    'Add "object1" attribute to Link ClassBehavior

    Set IxfAttribute = IxfClassBehavior.Attributes.Add("object1")
```

```

IxfAttribute.TypeDefinition.ValueType = dtObjectReference

IxfAttribute.Required = True

'Add "object2" attribute to Link ClassBehavior
Set IxfAttribute = IxfClassBehavior.Attributes.Add("object2")

IxfAttribute.TypeDefinition.ValueType = dtObjectReference

IxfAttribute.Required = True

End Sub

```

To add Directed Link, Tree Link, and TimeStamp Class Behaviors, see source files.

Execute Functions

```

Public Sub AddClassBehaviorsDefinitionsToSchema(Schema As ISmIxfSchema,
SchemaHelper As ISmIxfSchemaHelper)

    AddAPISupportedClassBehaviors Schema, SchemaHelper

    AddMessageClassBehavior Schema

    AddFolderClassBehavior Schema

    AddLinkClassBehavior Schema

    AddDirectedLinkClassBehavior Schema

    AddTreeLinkClassBehavior Schema

End Sub

```

Adding Classes

The following functions add the required classes to the schema:

Add Message Class

```

Private Sub AddMessageClass(Schema As ISmIxfSchema, SchemaHelper As
ISmIxfSchemaHelper)

    Dim IxfClass As ISmIxfClass

    Dim IxfClassBehavior As ISmIxfClassBehavior

```

```
'Add class "message" to Schema Classes
Set IxfClass = Schema.Classes.Add("message")

'Enable IXF TimeStamp Standard Behavior for message class as required
'by DB_MESSAGING_URI Domain Behavior
SchemaHelper.EnableTimeStampForClass IxfClass

'Declare CB_MESSAGE_URI ClassBehavior in message class as required
'by DB_MESSAGING_URI Domain Behavior
Set IxfClassBehavior = Schema.ClassesBehaviors.ItemByURI(CB_MESSAGE_URI)

IxfClass.CurrentClassBehaviors.Add IxfClassBehavior
End Sub
```

Add Folder Class

```
Private Sub AddFolderClass(Schema As ISmIxfSchema, SchemaHelper As
ISmIxfSchemaHelper)

    Dim IxfClass As ISmIxfClass

    Dim IxfClassBehavior As ISmIxfClassBehavior

    'Add class "folder" to Schema Classes
    Set IxfClass = Schema.Classes.Add("folder")

    'Enable IXF TimeStamp Standard Behavior for folder class as required
    'by DB_MESSAGING_URI Domain Behavior
    SchemaHelper.EnableTimeStampForClass IxfClass

    'Declare CB_FOLDER_URI ClassBehavior in folder class as required
    'by DB_MESSAGING_URI Domain Behavior
```

```

        Set IxfClassBehavior = Schema.ClassesBehaviors.ItemByURI(CB_FOLDER_URI)

        IxfClass.CurrentClassBehaviors.Add IxfClassBehavior
    End Sub

```

Add FolderLink Class

```

Private Sub AddFolderLinkClass(Schema As ISmIxfSchema)

    Dim IxfClass As ISmIxfClass

    Dim IxfClassBehavior As ISmIxfClassBehavior

    'Add class "folderLink" to Schema Classes
    Set IxfClass = Schema.Classes.Add("folderLink")

    'Declare CB_LINK_URI, CB_DIRECTEDLINK_URI, and CB_TREELINK_URI
    'ClassBehaviors in folderLink class as required
    'by DB_MESSAGING_URI Domain Behavior. When writing the object, only
    'the CB_LINK_URI is used. (The presence of the CB_DIRECTEDLINK_URI,
    'and CB_TREELINK_URI ClassBehaviors cause the CB_LINK_URI to be
    'interpreted as a directed parent-son tree link.)

    Set IxfClassBehavior = Schema.ClassesBehaviors.ItemByURI(CB_LINK_URI)

    IxfClass.CurrentClassBehaviors.Add IxfClassBehavior

    Set IxfClassBehavior =
    Schema.ClassesBehaviors.ItemByURI(CB_DIRECTEDLINK_URI)

    IxfClass.CurrentClassBehaviors.Add IxfClassBehavior

    Set IxfClassBehavior = Schema.ClassesBehaviors.ItemByURI(CB_TREELINK_URI)

    IxfClass.CurrentClassBehaviors.Add IxfClassBehavior
End Sub

```

Add Attachment Class

```
Private Sub AddAttachmentClass(Schema As ISmIxfSchema, SchemaHelper As
ISmIxfSchemaHelper)

    Dim IxfClass As ISmIxfClass

    'Add class "attachment" to Schema Classes

    Set IxfClass = Schema.Classes.Add("attachment")

    'Enable IXF FileAssociation Standard Behavior for folder class as
    'required by DB_MESSAGING_URI Domain Behavior

    SchemaHelper.EnableFileAssociationForClass IxfClass

End Sub
```

Add MessageAttachmentLink Class

```
Private Sub AddMessageAttachmentLinkClass(Schema As ISmIxfSchema)

    Dim IxfClass As ISmIxfClass

    Dim IxfClassBehavior As ISmIxfClassBehavior

    'Add class "messageAttachmentLink" to Schema Classes

    Set IxfClass = Schema.Classes.Add("messageAttachmentLink")

    'Declare CB_LINK_URI, and CB_DIRECTEDLINK_URI ClassBehaviors in
    'messageAttachmentLink class as required by DB_MESSAGING_URI Domain
    'Behavior

    Set IxfClassBehavior = Schema.ClassesBehaviors.ItemByURI(CB_LINK_URI)

    IxfClass.CurrentClassBehaviors.Add IxfClassBehavior

    Set IxfClassBehavior =
Schema.ClassesBehaviors.ItemByURI(CB_DIRECTEDLINK_URI)

    IxfClass.CurrentClassBehaviors.Add IxfClassBehavior

End Sub
```

Add FolderMessageLink Class

```
Private Sub AddFolderMessageLinkClass(Schema As ISmIxfSchema)

    Dim IxfClass As ISmIxfClass

    Dim IxfClassBehavior As ISmIxfClassBehavior

    'Add class "folderMessageLink" to Schema Classes
    `Set IxfClass = Schema.Classes.Add("folderMessageLink")

    'Declare CB_LINK_URI, and CB_DIRECTEDLINK_URI ClassBehaviors in
    'folderMessageLink class as required by DB_MESSAGING_URI Domain
    'Behavior

    Set IxfClassBehavior = Schema.ClassesBehaviors.ItemByURI(CB_LINK_URI)

    IxfClass.CurrentClassBehaviors.Add IxfClassBehavior

    Set IxfClassBehavior =
    Schema.ClassesBehaviors.ItemByURI(CB_DIRECTEDLINK_URI)

    IxfClass.CurrentClassBehaviors.Add IxfClassBehavior

End Sub
```

Execute Functions

```
Public Sub AddClassesDefininitionsToSchema(Schema As ISmIxfSchema,
SchemaHelper As ISmIxfSchemaHelper)

    AddMessageClass Schema, SchemaHelper

    AddFolderClass Schema, SchemaHelper

    AddFolderLinkClass Schema

    AddAttachmentClass Schema, SchemaHelper

    AddMessageAttachmentLinkClass Schema

    AddFolderMessageLinkClass Schema

End Sub
```

Adding Domain Behaviors

The following functions add the required Domain Behavior to the schema:

Add Messaging Domain Behavior

```
Private Sub AddMessagingDomainBehavior(Schema As ISmIxfSchema)

    Dim IxfDomainBehavior As ISmIxfDomainBehavior

    Dim IxfClass As ISmIxfClass

    'Add Domain Behavior DB_MESSAGING_URI" to Schema DomainBehaviors
    Set IxfDomainBehavior = Schema.DomainBehaviors.Add(DB_MESSAGING_URI)

    'Assign the Domain Behavior Roles to their corresponding classes
    Set IxfClass = Schema.Classes.ItemByName("message")
    IxfDomainBehavior.RoleClassMapping("message") = IxfClass

    Set IxfClass = Schema.Classes.ItemByName("folder")
    IxfDomainBehavior.RoleClassMapping("folder") = IxfClass

    Set IxfClass = Schema.Classes.ItemByName("folderLink")
    IxfDomainBehavior.RoleClassMapping("folderLink") = IxfClass

    Set IxfClass = Schema.Classes.ItemByName("attachment")
    IxfDomainBehavior.RoleClassMapping("attachment") = IxfClass

    Set IxfClass = Schema.Classes.ItemByName("messageAttachmentLink")
    IxfDomainBehavior.RoleClassMapping("messageAttachmentLink") = IxfClass

End Sub
```

Execute Functions

```
Public Sub AddDomainBehaviorsToTheSchema(Schema As ISmIxfSchema)

    AddMessagingDomainBehavior Schema

End Sub
```

Writing the Data

This section shows how to write the data to a data file. Two types of objects are written:

Basic Objects

Link Objects

Basic Objects

Write Message Object

```
Private Sub CreateMessageObject(DataWriter As ISmIxfDataWriter, WriterHelper
As ISmIxfWriterHelper)

    Dim IxfObject As ISmIxfObject

    Dim BehaviorValues As ISmIxfAttributesValues

    Dim TimeStamp As ISmIxfTimeStamp

    'Instantiate an object from the message class; give it an Id
    Set IxfObject = DataWriter.ObjectWriter.NewObject("message", "OID_1")

    'Assign values to CB_MESSAGE_URI BehaviorValues
    Set BehaviorValues = IxfObject.GetBehaviorValues(CB_MESSAGE_URI)
    BehaviorValues.Item("from") = "Bruce Mayer"
    BehaviorValues.Item("to") = "David Stein"
    BehaviorValues.Item("subject") = " The Sample1.doc file "
    BehaviorValues.Item("body") = " Attached is the Sample1.doc file"

    'TimeStamp the message object
    Set TimeStamp = WriterHelper.TimeStampWriter.CastToTimeStamp(IxfObject)

    TimeStamp.CreationTime = Now
```



```
        'Save the message object to the data file
        IxfObject.Save
    End Sub
```

Create and Embed the File Object

```
Private Sub CreateFileObjectAndEmbedFile(WriterHelper As ISmIxfWriterHelper)

    Dim IxfFile As ISmIxfFile

    'Instantiate a File object and give it values
    Set IxfFile = WriterHelper.FileWriter.NewFile("OID_2")
    IxfFile.FileName = " Sample1.doc"
    IxfFile.ContentType = "text/xml"
    IxfFile.SetSource ("Sample1.doc")

    'Embed the File object and save it to the data file
    WriterHelper.FileWriter.EmbedFile IxfFile
    IxfFile.Save
End Sub
```

Write Attachment Object

```
Private Sub CreateAttachmentObject(DataWriter As ISmIxfDataWriter,
WriterHelper As ISmIxfWriterHelper)

    Dim IxfObject As ISmIxfObject

    Dim FileAssociation As ISmIxfFileAssociation

    'Instantiate an object from the attachment class; give it an Id
    Set IxfObject = DataWriter.ObjectWriter.NewObject("attachment", "OID_3")
```

```

        'Associate the File object OID_2 with the attachment object

        Set FileAssociation =
WriterHelper.FileWriter.CastToFileAssociation(IxfObject)

        FileAssociation.FileId = "OID_2"

        'Save the attachment object

        IxfObject.Save

End Sub

```

Write Folder Objects

```

Private Sub CreateFolderObjects(DataWriter As ISmIxfDataWriter, WriterHelper
As ISmIxfWriterHelper)

    Dim IxfObject As ISmIxfObject

    Dim BehaviorValues As ISmIxfAttributesValues

    Dim TimeStamp As ISmIxfTimeStamp

    'Development folder

    'Instantiate an object from the folder class; give it an Id

    Set IxfObject = DataWriter.ObjectWriter.NewObject("folder", "OID_5")

    'Get CB_FOLDER_URI ClassBehavior BehaviorValues for this folder 'object

    Set BehaviorValues = IxfObject.GetBehaviorValues(CB_FOLDER_URI)

    'Name the folder "Development"

    BehaviorValues.Item("name") = "Development"

    'Time-stamp the "Development" folder

    Set TimeStamp = WriterHelper.TimeStampWriter.CastToTimeStamp(IxfObject)

    TimeStamp.CreationTime = Now

```

```
'Save the "Development" folder
IxfObject.Save

'ixf folder
'Instantiate another object from the folder class; give it an Id
Set IxfObject = DataWriter.ObjectWriter.NewObject("folder", "OID_6")

'Get CB_FOLDER_URI ClassBehavior BehaviorValues for this folder 'object and
name it "ixf"

Set BehaviorValues = IxfObject.GetBehaviorValues(CB_FOLDER_URI)
BehaviorValues.Item("name") = "ixf"

'Time-stamp the "ixf" folder
Set TimeStamp = WriterHelper.TimeStampWriter.CastToTimeStamp(IxfObject)

TimeStamp.CreationTime = Now

'Save the "ixf" folder
IxfObject.Save

'Sample1 folder
'Instantiate another object from the folder class; give it an Id
Set IxfObject = DataWriter.ObjectWriter.NewObject("folder", "OID_7")

'Get CB_FOLDER_URI ClassBehavior BehaviorValues for this folder 'object and
name it "Sample1"

Set BehaviorValues = IxfObject.GetBehaviorValues(CB_FOLDER_URI)
BehaviorValues.Item("name") = "Sample1"
```

```

        'Time-stamp the "Sample1" folder
        Set TimeStamp = WriterHelper.TimeStampWriter.CastToTimeStamp(IxfObject)

        TimeStamp.CreationTime = Now

        'Save the "Sample1" folder
        IxfObject.

    End Sub

```

Link Objects

Write MessageAttachmentLink Object

```

Private Sub CreateMessageAttachmentLinkObject(DataWriter As ISmIxfDataWriter)

    Dim IxfObject As ISmIxfObject

    Dim BehaviorValues As ISmIxfAttributesValues

    'Instantiate an object from the messageAttachmentLink class;
    'give it an Id
    Set IxfObject = DataWriter.ObjectWriter.NewObject("messageAttachmentLink",
"OID_4")

    'Get CB_LINK_URI ClassBehavior BehaviorValues for this object
    Set BehaviorValues = IxfObject.GetBehaviorValues(CB_LINK_URI)

    'Link between the message object and the attachment object
    BehaviorValues.Item("object1") = "OID_1"
    BehaviorValues.Item("object2") = "OID_3"

    'Save the messageAttachmentLink object
    IxfObject.Save

End Sub

```

Write FolderLink Objects

```
Private Sub CreateFolderLinkObjects(DataWriter As ISmIxfDataWriter)

    Dim IxfObject As ISmIxfObject

    Dim BehaviorValues As ISmIxfAttributesValues

    'Link "Development" folder as parent of "iXF" folder

    Set IxfObject = DataWriter.ObjectWriter.NewObject("folderLink", "OID_8")

    Set BehaviorValues = IxfObject.GetBehaviorValues(CB_LINK_URI)

    BehaviorValues.Item("object1") = "OID_5"
    BehaviorValues.Item("object2") = "OID_6"

    'Save the folderLink

    IxfObject.Save

    'Link "iXF" folder as parent of "Sample1" folder

    Set IxfObject = DataWriter.ObjectWriter.NewObject("folderLink", "OID_9")

    Set BehaviorValues = IxfObject.GetBehaviorValues(CB_LINK_URI)

    BehaviorValues.Item("object1") = "OID_6"
    BehaviorValues.Item("object2") = "OID_7"

    'Save the folderLink

    IxfObject.Save

End Sub
```

Write FolderMessageAttachment Objects

```
Private Sub CreateFolderMessageAttachmentObjects(DataWriter As
ISmIxfDataWriter)
```

```

Dim IxfObject As ISmIxfObject

Dim BehaviorValues As ISmIxfAttributesValues

'Link where "Sample1" folder as parent of the message with the
'subject "The Sample1.doc file"

Set IxfObject = DataWriter.ObjectWriter.NewObject("folderMessageLink",
"OID_8")

Set BehaviorValues = IxfObject.GetBehaviorValues(CB_LINK_URI)

BehaviorValues.Item("object1") = "OID_7"

BehaviorValues.Item("object2") = "OID_1"

'Save the link

IxfObject.Save

End Sub

```

Execute Functions

```

Public Sub CreateData(Writer As ISmIxfWriter, StdHelper As SmIxfStdHelper)

Dim WriterHelper As ISmIxfWriterHelper

Dim DataWriter As ISmIxfDataWriter

Set WriterHelper = StdHelper.CreateWriterHelper(Writer)

CreateMessageObject Writer.DataWriter, WriterHelper

CreateFileObjectAndEmbedFile WriterHelper

CreateAttachmentObject Writer.DataWriter, WriterHelper

CreateMessageAttachmentLinkObject Writer.DataWriter

CreateFolderObjects Writer.DataWriter, WriterHelper

CreateFolderLinkObjects Writer.DataWriter

CreateFolderMessageAttacmentObjects Writer.DataWriter

```

```
End Sub
```

Reading the Data

This section shows how to use the Reader and the ReaderHelper to read the data file.

Read Data Objects

```
Public Sub ReadData(Reader As ISmIxfReader, StdHelper As SmIxfStdHelper)

    Dim ReaderHelper As ISmIxfReaderHelper

    Dim ObjectIterator As ISmIxfObjectIterator

    Dim IxfObject As ISmIxfObject

    Dim IxfFile As ISmIxfFile

    Set ReaderHelper = StdHelper.CreateReaderHelper(Reader)

    Set ObjectIterator = Reader.DataReader.ObjectReader.GetObjectIterator

    While ObjectIterator.AtEnd = False

        Set IxfObject = ObjectIterator.GetObject

        'Read in object

        HandleObject IxfObject, Reader

        'See if it is File object

        Set IxfFile = ReaderHelper.FileReader.CastToFile(IxfObject)

        If Not (IxfFile Is Nothing) Then

            HandleFileObject IxfFile

        End If

        ObjectIterator.Next

    Wend

End Sub
```

Handle a File Object

```
Private Sub HandleFileObject(IxfFile As ISmIxfFile)

    If IxfFile.Embedded = True Then

        IxfFile.Extract App.Path + "\ExtractedFiles"

    End If

End Sub
```

Read an Object

```
Private Sub HandleObject(IxfObject As ISmIxfObject, IxfReader As SmIxfReader)

    Dim IxfClass As ISmIxfClass

    Dim IxfAttribute As ISmIxfAttribute

    Dim IxfClassBehavior As ISmIxfClassBehavior

    Dim Value As Variant

    Dim Values As ISmIxfAttributesValues

    Dim i, j As Integer

    Set IxfClass = IxfReader.Schema.Classes.ItemByName(IxfObject.ixfClassName)

    frmSample1.lstObjects.ListItems.Add , , "Object Id = " + IxfObject.Id +
    ":"

    frmSample1.lstObjects.ListItems.Add , , " Class = " + IxfClass.Name

    'class attributes

    If IxfClass.AllAttributes.Count > 0 Then

        frmSample1.lstObjects.ListItems.Add , , " Attributes:"

        Set Values = IxfObject.Values

        For i = 0 To IxfClass.AllAttributes.Count - 1
```



```

        Set IxfAttribute = IxfClass.AllAttributes.Item(i)

        Set Value = Values.Item(IxfAttribute.Name)

        If Not VarType(Value) = vbNull Then

            frmSample1.lstObjects.ListItems.Add , , " " +
IxfAttribute.Name + " = " + Value

        End If

    Next

End If

'Behavior attributes
If IxfClass.AllBehaviors.Count > 0 Then

    frmSample1.lstObjects.ListItems.Add , , " BehaviorAttributes:"

    For i = 0 To IxfClass.AllBehaviors.Count - 1

        Set IxfClassBehavior = IxfClass.AllBehaviors.Item(i)

        Set Values = IxfObject.GetBehaviorValues(IxfClassBehavior.URI)

        For j = 0 To IxfClassBehavior.Attributes.Count - 1

            Set IxfAttribute = IxfClassBehavior.Attributes.Item(j)

            Value = Values.Item(IxfAttribute.Name)

            If Not VarType(Value) = vbNull Then

                frmSample1.lstObjects.ListItems.Add , , " " +
IxfAttribute.Name + " = " + CStr(Value)

            End If

        Next

    Next

End If

frmSample1.lstObjects.ListItems.Add , , ""

End Sub

```

Executing the Application

This section shows how to execute the application.

```
Private Sub cmdCreateIXF_Click()  
  
    Dim IxfWriter As SmIxfWriter  
  
    Dim IxfStdHelper As SmIxfStdHelper  
  
    Dim IxfFileName As String  
  
  
    lstObjects.ListItems.Clear  
  
  
    dlgIxfFile.ShowOpen  
  
    IxfFileName = dlgIxfFile.FileName  
  
    If IxfFileName = vbNullString Then Exit Sub  
  
  
    Set IxfWriter = CreateObject("SmartIXF1.SmIxfWriter")  
    Set IxfStdHelper = CreateObject("SmartIXF1.SmIxfStdHelper")  
  
  
    If btnCreateSchema.Value = True Then  
        IxfWriter.SetSchemaMode mtEmbedded  
  
        SchemaCreation.CreateSchema IxfWriter.Schema, IxfStdHelper  
    Else  
        IxfWriter.SetSchemaMode mtExternal, "http://example.com/messaging",  
        "sample1.xsd", True  
    End If  
  
  
    IxfWriter.CreateIxfArchiveFile IxfFileName  
  
    DataWriting.CreateData IxfWriter, IxfStdHelper  
  
    IxfWriter.CloseIxfArchiveFile
```

```
        MsgBox ("Done")

End Sub

Private Sub cmdReadIxf_Click()

    Dim IxfReader As SmIxfReader

    Dim IxfStdHelper As SmIxfStdHelper

    Dim IxfFileName As String

    lstObjects.ListItems.Clear

    dlgIxfFile.ShowOpen

    IxfFileName = dlgIxfFile.FileName

    If IxfFileName = vbNullString Then Exit Sub

    Set IxfReader = CreateObject("SmartIXF1.SmIxfReader")

    Set IxfStdHelper = CreateObject("SmartIXF1.SmIxfStdHelper")

    IxfReader.OpenIxfArchiveFile IxfFileName, True

    DataReading.ReadData IxfReader, IxfStdHelper

    IxfReader.Close

    MsgBox ("Done")

End Sub
```

12. SmarTeam Client Libraries Overview

This chapter contains a brief overview of the **SmarTeam** Client libraries described in this document.

- SmartClientContext Library
- SmartClientContextService Library
- SmartClientServices Library
- SmartClientConfiguration Library
- SmartInet Library
- SmartFileCatalog Library
- SmartRecordList Library
- SmartIntegrationServices Library
- SmartGUIServices Library
- SmartEmbeddedScripts Library

SmartClientContext Library

Provides access to client libraries.

ISmClientContext

If you are using SmarTeam – Web Editor, SmartClientContext gives you access to all client libraries.

Methods

The SmartClientContext object has the following methods:

Method	Description
ClientServices	Accesses ClientServices. Returns ISmClientServices.
FileCatalog	Accesses FileCatalog. Returns ISmFileCatalog
EmbeddedScripts	Accesses EmbeddedScripts. Returns ISmEmbeddedScripts
GuiServices	Accesses GuiServices. Returns ISmGuiServices
Initialize (ApplicationName)	Initializes specified application for the Client Libraries.
IntegrationService	Accesses IntegrationService. Returns ISmIntegrationServices
ClientConfiguration	Accesses ClientConfiguration. Returns ISmClientConfiguration

Obtaining the SmartClientContext Object

```
clientContext = SmCreateSmClientContext();
```

Examples of these methods appear in each Client Library.

SmartClientContextService Library

Provides access to client libraries.

ISmClientContextService

This service is used only with SmarTeam – Editor and is only used there to create an instance of the File Catalog library.

Properties

The SmartClientContextService object contains the following properties:

Property	Description
ClientContext	Entry point for all Client libraries
Workspaces	If the database includes the mechanism of collaborative design, returns a SmarTeam RecordList containing a list of all Workspaces the database associated with the session.
IsSharedMode	Returns true if the session database includes the mechanism of collaborative design

Examples of these properties appear in the File Catalog library.

SmartClientServices Library

This service handles client library services.

ISmClientServices

The ISmClientServices object provides methods to create and manage a ISmClientDictionary.

Properties

The ISmClientServices object has the following properties:

Property	Description
UserId	User name
DatabaseId	Id of current database
ReplicaId	Replica Id of current database

Methods

The ISmClientServices object has the following methods:

Method	Description
CreateDictionary	Creates a dictionary. Returns ISmClientDictionary.

ISmClientDictionary

The ISmClientDictionary object represents a client dictionary.

Properties

The ISmClientDictionary object has the following properties:

Property	Description
Group	Returns a group of the dictionary by keyname. ISmDictionaryProp

Methods

The ISmClientDictionary object has the following methods:

Method	Description
CreateDictionaryGroup	Creates a dictionary group. Returns ISmDictionaryProperty.

ISmDictionaryGroup

The ISmDictionaryGroup object represents a dictionary group.

Properties

The ISmDictionaryGroup object has the following properties:

Property	Description
DictionaryProperty	Returns a dictionary property by name. ISmDictionaryProperty

Methods

The ISmDictionaryGroup object has the following methods:

Method	Description
CreateDictionaryProperty:	Creates a dictionary property. Returns ISmDictionaryProperty.

ISmDictionaryProperty

The ISmDictionaryProperty object represents a dictionary group property.

Properties

The ISmDictionaryProperty object has the following properties:

Property	Description
Value	Value of the dictionary property.

SmartClientConfiguration Library

This service handles client library configuration.

ISmClientConfiguration

The ISmClientConfiguration object provides methods to access client configuration items

Methods

The ISmClientConfiguration object has the following methods:

Method	Description
GetValue	Gets the value of a configuration item.
SetValue	Sets the value of a configuration item
NewSmConfiguration ValueList	Creates a new object of type SmConfigurationValueList
GetValueList	Gets the collection of all configuration item values corresponding to a single key. Multiple identical values are entered as a single item in the collection.
SetValueList	Sets the collection of all configuration item values corresponding to a single key

Examples

Get a configuration item value (GetValue)

```
Set value = SmClientConfiguration.GetValue("Pure Client Configuration",  
"server")
```

Set a configuration item value (SetValue)

```
SmClientConfiguration.SetValue "smarteam.std.clientLibraries.UI",  
"smarteam.std.clientLibraries.ui.insert",  
/WebEditor/Views/Searches/Default.aspx
```

Get and set a value list (GetValueList, SetValueList)

```
' GetValueList
```

```
Set SmConfigurationValueList =  
SmClientConfiguration.GetValueList("smarteam.std.clientLibraries.UI",  
"smarteam.std.clientLibraries.ui.insert")
```

```
' SetValueList

Set SmValueList = SmClientConfiguration.NewSmConfigurationValueList;

SmValueList.Add "value1"

SmValueList.Add "value2"

SmValueList.Add "value3"

SmClientConfiguration.SetValueList "smarteam.std.clientLibraries.UI",
"smarteam.std.clientLibraries.ui.insert", SmValueList
```

ISmConfigurationValueList

The ISmConfigurationValueList object represents a collection of configuration item values.

Properties

The ISmConfigurationValueList object has the following properties:

Property	Description
Item	Gets a value from the collection by index

Methods

The ISmConfigurationValueList object has the following methods:

Method	Description
Add	Adds a value to the collection. Returns the index of the added value.
Remove	Removes a value from the collection by index
IndexOf	Gets index of a configuration item value

SmartInet Library

This service handles low level connection to the server.

IHttpConnection

The IHttpConnection object provides methods for connecting to the server

Methods

The IHttpConnection object has the following methods:

Method	Description
Open	Opens a connection to the server.
Close	Closes a connection to the server
CreateContext	Creates a session on the server. Returns IHttpContext

IHttpContext

The IHttpContext object provides methods for opening a session on the server.

Methods

The IHttpContext object has the following methods:

Method	Description
Open	
RequestAddHeader	
RequestSubmit	
RequestGetHeader	
ResponseRead	
ResponseSaveToFile	
RequestUploadFile	
Close	

IHttpUtils

The IHttpUtils object provides utilities for working on the server.

Methods

The IHttpUtils object has the following methods:

Method	Description
SaveUrlToFile	Save URL to file
DetectFileMimeType	Detects file Mime type

SmartFileCatalog Library

The SmartFileCatalog library comprises objects that enable the following functionality:

Client-side file management for running SmarTeam in all possible configurations

Provides API support for integrations, life-cycle operations, SmarTeam File Explorer, and collaborative design

Folder-based structure management for file storage on client's machine or in network location

Provides a mechanism for accessing and updating file object attributes

See Chapter [13](#) for further details.

SmartRecordList Library

The SmartRecordList library comprises objects that enable the following functionality:

Allows a client to work with record list data objects that are similar to those in the SmarTeam API

See Chapter [14](#) for further details.

SmartIntegrationServices Library

The SmartIntegrationServices library comprises objects that enable the following functionality:

- Exposes SmarTeam functionality in Host Applications
- Implements mapping of component data to SmarTeam objects
- Implements common integration tasks
- Implements data transfer to SmarTeam Database
- Provides a multi-platform solution
- Components tree traversal
- Optimized data exchanges

See SmarTeam Integration Tools Library on page [19](#) for further details.

SmartGUIServices Library

The SmartGUIServices library comprises objects that enable the following functionality:

Displays standard SmarTeam – WebEditor GUI windows in client applications

Enables selection of browser to be used in the window

Receives user input and initiates appropriate action at client

See SmarTeam GUI Services Library on page [17](#) for further details.

SmartEmbeddedScripts Library

The SmartEmbeddedScripts library comprises objects that enable the following functionality:

Send scripts to the server for execution

Receive results as various types, including Record List

Upload and download files from the server.

See Chapter [17](#) for further details.

13. SmartFileCatalog Library

General Description

The **SmarTeam** File Catalog library comprises objects that enable the following functionality:

- Client-side file management for running SmarTeam in all possible configurations
- Provides API support for integrations, life-cycle operations, SmarTeam File Explorer, and collaborative design
- Folder-based structure management for file storage on client's machine or in network location
- Provides a mechanism for accessing and updating file object attributes

Dependencies

The **SmarTeam** File Catalog library has the following dependencies:

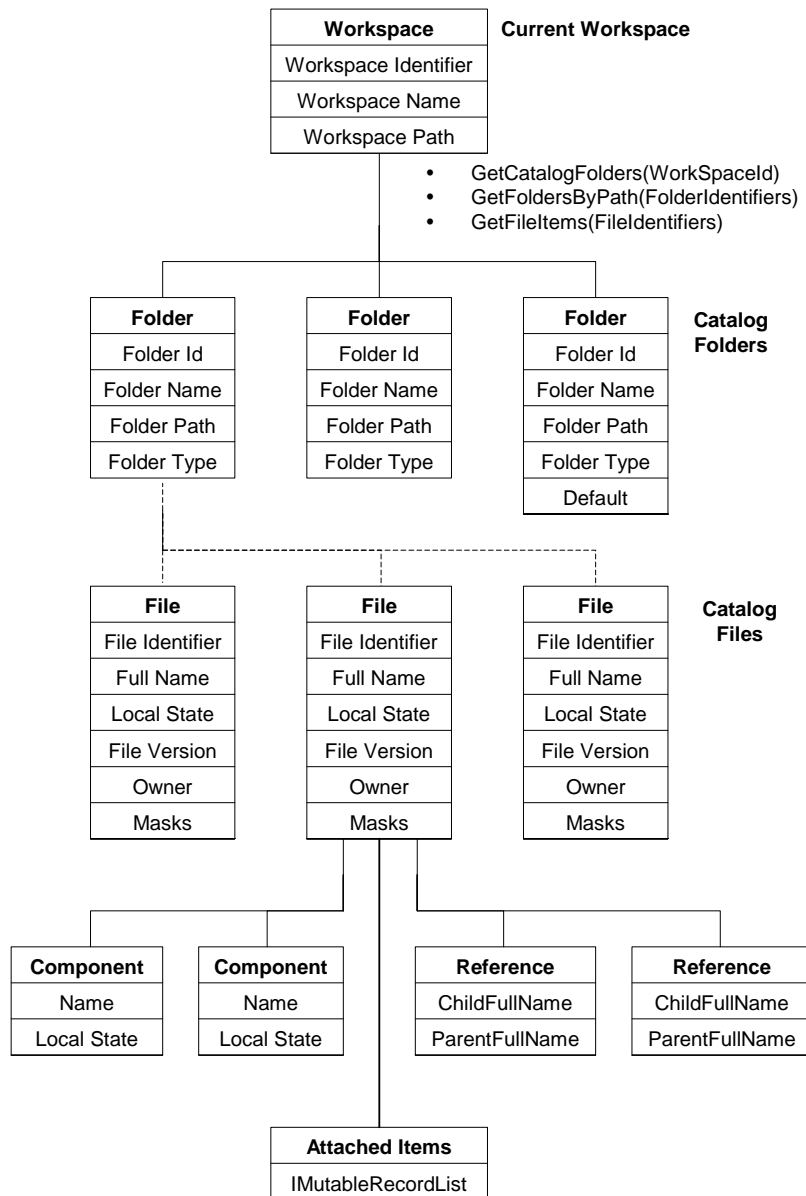
- **SmarTeam** Record List library
- **SmarTeam** Client Services.

Overview of File Catalog Library

The File Catalog manages the files and data connected with a specific database. In a SmarTeam session, the File Catalog relates to the information from the connected database for that session.

File Catalog Object Organization

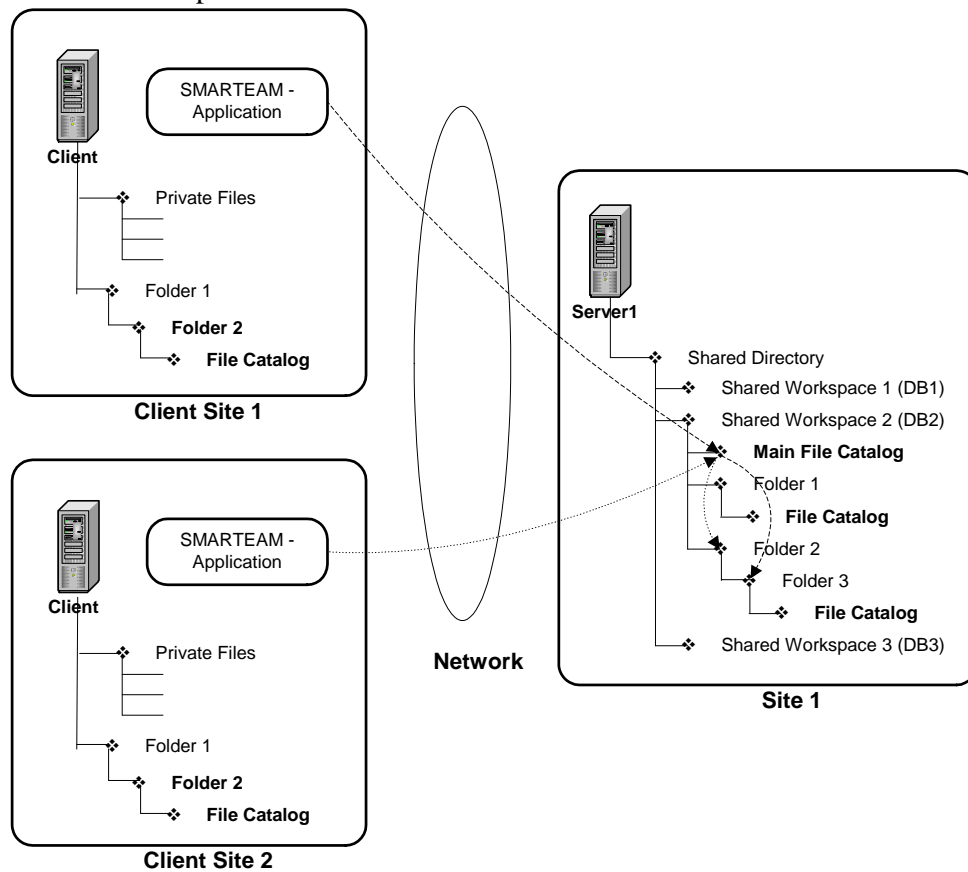
The following diagram shows the organization of the File Catalog objects. Note that, for reasons of efficiency, the SmFile objects are not directly accessed through the SmFolder objects. Instead, they are fetched at run time by search functions like GetFileitems.



File Catalog in a Shared Workspace

The File Catalog provides file management abilities for working with both private user files and with files in a Shared Workspace. “Private user files” refers to a situation where an individual engineer copies files from a vault and edits or views the files in a non-shared working directory. A Shared Workspace refers to a situation where team members work on files in a *shared workspace*, referencing each other’s designs from the shared location.

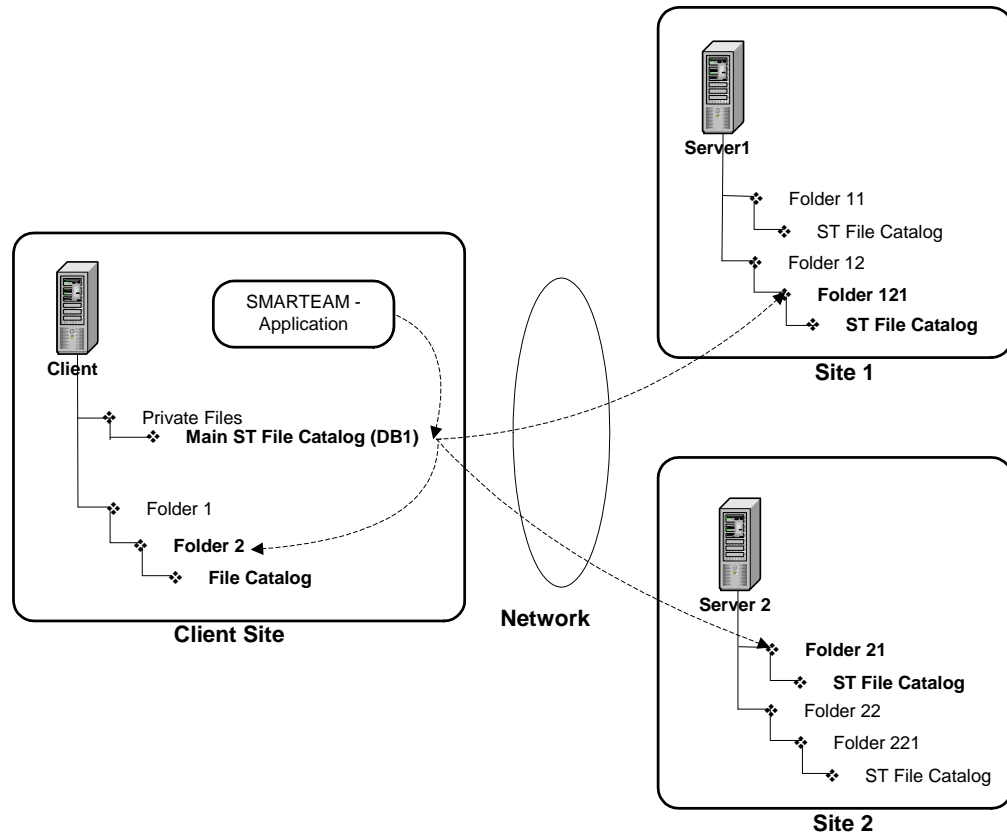
The figure below shows how two users use the File Catalog to access files in a Shared Workspace.



Catalog File Access in a Shared Workspace

File Catalog with Private Files

The figure below shows how the File Catalog is used to access Private files.

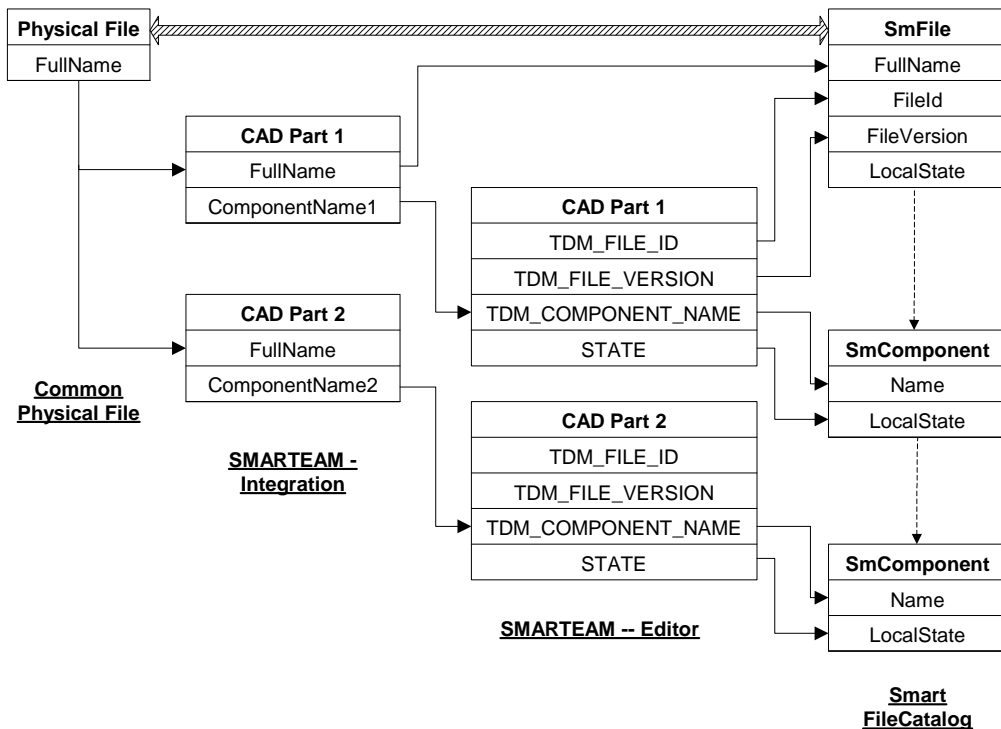


Using the File Catalog to Access Private Files

Relation to SmarTeam Processes

When using the File Catalog with the SmarTeam – Editor, the registration of a file object in the File Catalog represents a physical file that is associated with a file-managed SmarTeam object, for example, a CAD Part, as shown below. The registration is performed when the physical file is moved or copied from its vault, for example, under a Check Out or View operation on the SmarTeam object. When the file is returned to the vault, the physical file and its registration in the File Catalog object are deleted, unless a copy of the file is left out of vault.

The following figure shows the relation between the checked-out file-managed object CAD Part 1 and the file registration created in the File Catalog for the checked-out file. The figure also shows a CFO CAD Part 2 that refers to the same physical file. The relation between the two CFO's is represented in the FileCatalog by assigning both CAD Part 1 and CAD Part 2 as component objects of the file object. One of the checked-out components is designated as the primary component of the file object.



SmFile Attributes

The information in the File Catalog corresponds with the information in SmarTeam – Editor, as shown in the following table:

SmFile Attribute	Description	SmarTeam – Editor Object Attribute
FullName	Uniquely defines the SmFile object in the File Catalog	SmarTeam file-managed object does not use FullName to refer to the physical file.
FileId	Id of File. Should be same as TDM_FILE_ID attribute of SmarTeam file-managed object that refers to the file -- when that attribute exists.	TDM_FILE_ID is the identifier of the file referenced by the file-managed SmarTeam object. When a new SmarTeam file-managed object references a specific file name is created (for example, on CheckOut), then a new File Id is also created. SmFile.FileId uses that new value.
FileVersion	Version of file. Should be same as the TDM_FILE_VERSION attribute of the SmarTeam file-managed object that refers to the file	TDM_FILE_VERSION is a counter of version: the file referred to by the file-managed SmarTeam object. When operation Check-In is performed with replacement of previous revision, FileVersion increased by one.
Component Name	Name of component. Should be same as TDM_COMPONENT_NAME attribute of the SmarTeam file-managed object that refers to the file.	TDM_COMPONENT_NAME of the file-managed object.

Overview of Objects—ISmFileCatalog

This section presents a hierarchical overview of the main ISmFileCatalog objects including a description of the associated objects that are useful for the programmer:

The ISmFileCatalog is the highest-level object; its main purpose is to contain the other objects.

The major ISmFileCatalog components are shown in the following object diagram:

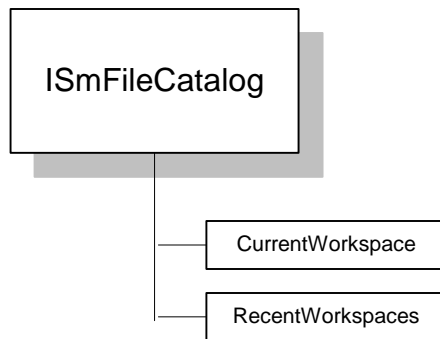


Figure 13-1 ISmFileCatalog Object Diagram

Properties

The ISmFileCatalog object contains the following properties:

Property	Description
CurrentWorkspace	Gets the GUID of the Current Workspace for private files and a shared workspace.
RecentWorkspaces	Retrieves a ISmWorkspaces collection that represents recently accessed workspaces. Returns ISmWorkspaces

Methods

The ISmFileCatalog object contains the following methods:

Method	Description
AddRecentWorkspace	Adds a specified Workspace to the recently accessed workspace collection SmRecentWorkspaces
DeleteFileFromCatalog	Deletes the SmFile object registration from File Catalog (the physical file is not deleted)
DeleteFilesFromCatalog	Deletes the registration of SmFile objects from File Catalog (the physical files are not deleted)
DeleteSmFile	Deletes the specified SmFile object from the File Catalog and deletes the physical file. Returns ISmResultItem
DeleteSmFiles	Deletes a set of specified SmFile objects from the File Catalog and deletes the physical files. Returns ISmResultItems
GetCatalogFolders	Gets File Catalog folders for a specified Workspace. It returns folders for either private files or shared Workspace. It does not return the Temporary folder for a shared Workspace. Returns SmFolders.
GetDefaultFolderOriginalPath	Gets the original (non-UNC) path of the default folder for a specified Workspace
GetDefaultFolderPath	Gets the path of the default Working Directory for a specified Workspace or for private files
GetFileItems	Gets information about SmFile objects in the File Catalog
GetFoldersByPath	Gets folders from SmCatalog for specified folder path string. Returns ISmFolders.
GetTemporaryFiles	Gets files from the Temporary folder for specified SmFile objects. Returns ISmResultItems.
GetTemporaryFolderOriginalPath	Gets the original (non-UNC) path of the Temporary Directory
GetTemporaryFolderPath	Gets the UNC path of the Temporary folder.
GetWorkspaceById	Gets Workspace by specified Workspace Id. Returns ISmWorkspace.

IsLinkAllowed	Returns True if an SmReference object can be established from a specified parent path to a specified child path. This method can return False when parent or child is not in the shared workspace.
IsOperationAllowed	Returns True if the specified operation is allowed for the specified path. This method can return False if the path is in the shared Workspace.
IsPartOfSharedWorkspace	Returns True if the specified path belongs to a directory tree of the shared workspace associated with the FileCatalog.
NewSmFileIdentifiers	Creates a new SmFileIdentifiers collection object. Returns ISmFileIdentifiers.
NewSmFiles	Creates a new SmFiles collection object. Returns ISmFiles.
NewSmFolderIdentifiers	Creates a new SmFolderIdentifiers collection object. Returns ISmFolderIdentifiers.
NewSmReferences	Creates a new SmReferences collection object. Returns ISmReferences.
NewSmRetrieveFilter	Creates a new SmRetrieveFilter object. Returns ISmRetrieveFilter.
RecentWorkspaces	Retrieves ISmWorkspaces collection that represents recently accessed workspaces. Returns ISmWorkspaces.
SetCurrentWorkspace	Sets the Workspace specified by its Id to be the CurrentWorkspace and changes the Default Working Directory to the Default Working Directory of newly set Workspace. If the Workspace Id is empty, it means that the user is working with private files.
SetDefaultFolderPath	Sets a folder specified by path and Workspace to be the Default Working Directory for the Workspace or for private files.
SetFileItemOwnership	Sets new ownership for a file specified by FullName.
SetTemporaryFolderPath	Sets a folder specified by path to be the Temporary Directory for this user.

Update	<p>Updates specified SmFile objects in File Catalog. Returns ISmResultItems.</p> <p>Links argument: The list of SmReferences to list in the FileCatalog. An SmReference in the Links parameter will be listed in the File Catalog by the Update method under the following circumstances: The SmFile object represented by the ParentName parameter of the SmReference object is included in the FileItems parameter of the Update method. The SmFile object represented by the ChildName of the SmReference object is included in the FileItems parameter of the Update method or is already listed in the FileCatalog.</p> <p>Otherwise the SmReference object is ignored.</p> <p>UpdateReferences argument When the Update References parameter is true the set of SmReferences in the FileCatalog is replaced by the set of SmReferences passed in the Links parameters. As a special case, when the set of SmReferences passed in the Links parameters is empty, the set of SmReferences in the FileCatalog is deleted. When the Update References parameter is false, the Links parameter is ignored and no change is made to the set of SmReferences in the File Catalog.</p> <p>DeleteFiles argument DeleteFiles controls whether FileCatalog is allowed to delete files during an Update if circumstances permit. If DeleteFiles is set to False, FileCatalog is prevented from deleting files during an Update under any circumstances. Default is True.</p>
--------	--

Obtaining the ISmFileCatalog Objects

You can access the File Catalog through the ISmFileCatalog object, which is accessible through the ClientContextService library.

```
Dim SmFileCatalog As SmartFileCatalog.ISmFileCatalog
```

```
Dim ClientContextService As SmartClientContextService.SmClientContextService
```

```
Set ClientContextService =
```

```
SmSession.GetService("SmartClientContextService.SmClientContextService")
```

```
Set SmFileCatalog = ClientContextService.ClientContext.FileCatalog
```

ISmFiles

The ISmFiles object represents a collection of ISmFile objects.

Properties

The ISmFiles object has the following properties:

Property	Description
Item	Returns a member of the collection by position. Returns ISm

Methods

The ISmFiles object has the following methods:

Method	Description
Add	Adds new SmFile object to the collection.
Remove	Removes a SmFile object from the collection
IndexOf	Returns the index of the first entry in the collection with a specified full file name

Example

Adds an SmFile object to the collection.

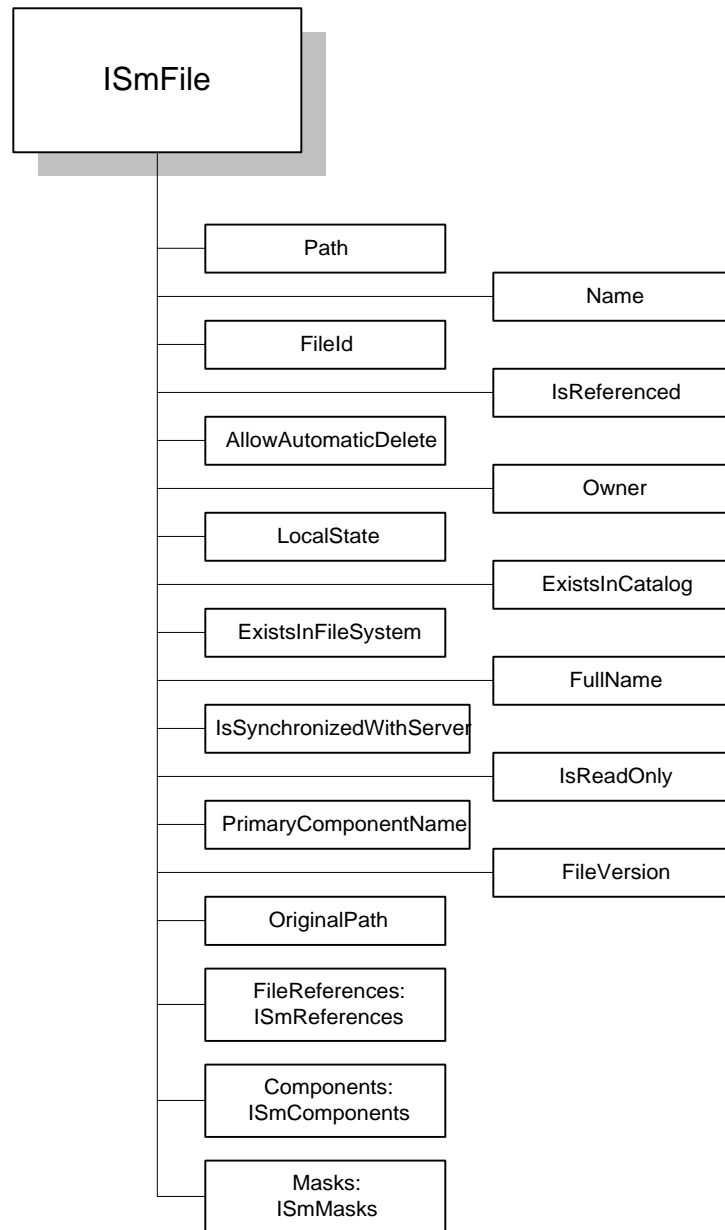
```
Dim SmFiles As SmartFileCatalog.ISmFiles  
  
Dim SmFile As SmartFileCatalog.ISmFile  
  
Set SmFiles = SmFileCatalog.NewSmFiles  
  
Set SmFile = SmFiles.Add("C:\Work\MyFile.bmp")
```

ISmFile

The SmFile object represents an individual file in the File Catalog.

A SmFile object represents a physical file in a folder and carries some of the physical file's attributes such as file name, path and read-only status.

The object diagram of ISmFile is shown below:

*Figure 13-2 ISmFile Object Diagram*

Properties

The ISmFile object has the following properties:

Property	Description
AllowAutomaticDelete	Returns or sets automatic delete for this SmFile. When true, Catalog can automatically delete the physical file and associated (mask) files when conditions permit. If false, a file can only be deleted by the DeleteSmFile method. Default is True.
Components	Retrieve collection object of ISmComponents. Returns ISmComponents
ExistsInCatalog	Returns true if this SmFile object exists in the File Catalog. Relevant on return from call to GetFileItems.
ExistsInFileSystem	Returns true, if a physical file exists that has this object's FullName. Relevant on return from call to GetFileItems.
FileId	Returns or sets the FileId of SmFile – should be as maintained in the TDM_FILE_ID attribute of the associated file-managed object in the SmarTeam database. The TDM_FILE_ID and TDM_FILE_VERSION attributes uniquely specify the File object in SmarTeam.
FileVersion	File version – should be as maintained in the TDM_FILE_VERSION attribute of the associated file-managed object in the SmarTeam database.
FullName	Returns the FullName (Path and FileName) of SmFile. The property uniquely specifies SmFile object in File Catalog.
IsReadOnly	Returns true on call to GetFileItems if the physical file is read-only on the file system
IsReferenced	Returns True if this SmFile object appears as a child in at least one SmReference object in the File Catalog. A value of True normally means that the File Catalog should delete the physical file from its working directory.

IsSynchronizedWithServer	<p>GetFileItems returns IsSynchronizedWithServer = True when modification date of the physical file corresponds with the SmFile modification date listed in the File Catalog.</p> <p><i>Description</i></p> <p>On registering a file in the File Catalog, (update operation), the modification date of the physical file is entered in the File Catalog.</p> <p>IsSynchronizedWithServer, as a returned value from GetFileItems, is calculated and read-only. GetFileItems looks at the current modification date of the physical file and compares with the file's modification date in the File Catalog. If it is the same, i.e. file was not changed, it returns IsSynchronizedWithServer = true. If it is different, it returns false, meaning that the file was modified. GetFileItem does not change the modification date registered in the File Catalog.</p>
LocalState	<p>Returns or sets the LocalState of SmFile as CatalogItemLocalStateEnum</p> <p><i>cilsUndefined</i> – undefined state in File Catalog. Also indicates "file not found" when returned by GetFileItems.</p> <p><i>cilsEditable</i> –equivalent to the Checked Out or New state of a file-managed SmarTeam object to which this object corresponds.</p> <p><i>cilsStandard</i> – for future use</p> <p><i>cilsNotEditable</i>- equivalent to the Copied or the View state of a file-managed SmarTeam object to which this object corresponds.</p>
Masks	Returns or sets the Masks collection of SmFile. Returns ISmMasks
Name	Returns the File Name (without path) of this SmFile
OriginalPath	Returns the local (non-UNC) path of SmFile. Use only for private files, not for a shared workspace.
Owner	Returns or sets the Owner of SmFile (same as SmFileCatalog.SetFileItemOwnership)
Path	Returns the path of the SmFile
PrimaryComponentName	<p>Returns or sets the PrimaryComponentName of the SmFile</p> <p>If there are no components other than this SmFile object itself, it is set as the primary component. If there are several CFO elements for this SmFile object, then the first to be registered in the File Catalog with its LocalState editable is defined as the primary component.</p>

Methods

The ISmFile object has the following methods:

Method	Description
GetAttachedItems	Retrieves AttachedItems attached to the object (not implemented). Specified by ItemType to transfer additional objects (as a Record List) related to the file
SetAttachedItems	Set AttachedItems to the object. Specified by ItemType and AttachedItems object
GetAttachedItem	Retrieves AttachedItem to the object. Specified by ItemType, Column name and AttachedItemId
SetAttachedItem	Set AttachedItem to the object. Specified by ItemType, Column name, AttachedItemId and ItemRecord
RemoveAttachedItems	Remove AttachedItems to the object. Specified by ItemType
RemoveAttachedItem	Remove AttachedItem to the object. Specified by ItemType, Column name and AttachedItemId
GetReferences	Retrieves collection object of SmReferences in the File Catalog for which this SmFile object is the referencing (parent) object. Returns ISmReferences
MarkToSynchronizeWithServer	Set the method argument to True to allow the SmFile modification date in the File Catalog to be updated on subsequent calls to the Update method. The Update method can change the modification date in the Catalog to correspond to the current modification date of the physical file. It will do so only if the method MarkToSynchronizeWithServer has been called with a value true previous to the call to Update. In the case that you only want to the Update method to update certain SmFile attributes in the File Catalog and not to update the SmFile modified status in the File Catalog, then do not call MarkToSynchronizeWithServer previous to calling the Update operation.

Obtaining the ISmFile Object

You obtain the ISmFile object as follows:

Example

The following:

```
Dim SmFiles As SmartFileCatalog.ISmFiles  
  
Dim SmFile As SmartFileCatalog.ISmFile  
  
Set SmFile = SmFiles.Add("C:\Work\MyFile.bmp")
```


ISmReferences

The ISmReferences object represents a collection of ISmReference objects.

Properties

The ISmReferences object has the following properties:

Property	Description
Item	Returns a member of the collection by position. Returns ISmReference.

Methods

The ISmReferences object has the following methods:

Method	Description
Add	Adds new SmReference object to the collection.
Remove	Removes a SmReference object from the collection

Example

Obtain an SmReferences collection object.

```
Dim SmReference As SmartFileCatalog.ISmReference

Dim SmReferences As SmartFileCatalog.ISmReferences

Set SmReferences = SmFileCatalog.NewSmReferences
```

ISmReference

The SmReference object represents a directed reference between two file objects for the purpose of controlling automatic deleting of files by the File Catalog.

An SmReference object is established between a referencing file object (called the parent file object), and a referenced file object (called the child file object), such as an assembly file and a part file in the assembly. The SmReference object expresses the fact that when the parent file is present in its working directory, then the child file should be present in its working directory.

As an example of when you would establish an SmReference object, SmReference objects might be established between file objects that have directed SmarTeam links between them and would normally need to be checked out together.

Automatic Delete of Files

The purpose of the File Catalog ISmReference is to provide a mechanism to insure that files are deleted from their working directories by the File Catalog automatic delete function at the proper time. The mechanism determines whether a child file is referenced by more than one parent file currently in work. It prevents automatically deleting a child file from its working directory on deletion of one parent file, when another parent still requires it to be there. Without this facility, a child file that is required by two separate parent files in work would be automatically deleted when one parent is checked in. Note that you can disable the File Catalog automatic delete facility for a specific file by setting SmFile.AllowAutomaticDelete to false.

Parent/Child Terminology

Note that the parent/child terminology is used with the SmReference object to emphasize that the direction of the reference is from the first parameter (referencing file) to the second parameter (referenced file) in the SmReferences.Add (ParentFullName, ChildFullName) method.

The parent/child terminology does not imply any connection to SmarTeam hierarchic links that may exist between the file objects and SmReference is also applicable to two file objects that are linked by a SmarTeam directed link.

Accessing a SmReference Object

You can access an SmReference object through its parent SmFile object. The GetFileReferences method of a SmFile object returns the all SmReference objects in which the SmFile object is a parent. You cannot access an SmReference through its child object.

Updating SmReference Object in the File Catalog

The Update method registers in the File Catalog the SmReference objects you have defined. It replaces SmReference objects in the File Catalog with the SmReference objects in the Links parameter. When the Links parameter is empty, all SmReference objects are deleted from the File Catalog. In order to avoid deleting all SmReference objects in the File Catalog when you pass an empty Links parameter, set the parameter UpdateReferences to False.

Properties

The ISmReference object has the following properties:

Property	Description
ParentFullName	Returns or sets the file name of the referencing file object SmReference.
ChildFullName	Returns or sets the file name of the referenced object SmReference.

Obtaining the ISmReference Object

You obtain the ISmReference object as follows:

```
Set SmReference = SmReferences.Add("C:\Work\Parent.sldasm",  
"C:\Work\Child.sldprt")
```

ISmComponents

The ISmComponents object represents a collection of ISmComponent objects representing SmarTeam CFO elements for a single SmFile object.

Properties

The ISmComponents object has the following properties:

Property	Description
Item	Returns a member of a collection by position. Returns ISmComponent.

Methods

The ISmComponents object has the following methods:

Method	Description
Add	Adds new SmComponent object to the collection.
Remove	Removes a SmComponent object from the collection.
IndexOf	Returns the index of the first entry in the collection with a specified Component name.

Example

Adds a SmComponent object to the collection.

```
Dim SmFile As SmartFileCatalog.ISmFile

Dim SmComponent As SmartFileCatalog.ISmComponent

Dim SmComponents As SmartFileCatalog.ISmComponents

Set SmComponents = SmFile.Components

Set SmComponent = SmComponents.Add("ComponentName")
```

ISmComponent

An SmComponent object represents a SmarTeam Component object in the File Catalog. The SmComponent.ComponentName property should be the same as the TDM_COMPONENT_NAME attribute of the SmarTeam file-managed object that refers to the file.

The ComponentName is used as the Component Identifier, for example:

```
ComponentIdentifiers.Add (ComponentName)
```

A SmComponent exists in the File Catalog when the corresponding Component File Catalog exists. The Component File Catalog, which contains the SmComponent properties, exists in a subdirectory of the File Catalog for its SmFile object.

You can determine if an SmComponent exists in the File Catalog by calling the GetFileItems method with its Component Name. If the Component exists in the File Catalog, the ExistsInCatalog property is set true on return. If the Component does not exist in the File Catalog, the LocalState parameter will be returned with the value *cilsUndefined*.

Properties

The ISmComponent object has the following properties:

Property	Description
Name	Returns or sets the ComponentName of SmComponent.
LocalState	Returns or sets the LocalState of SmComponent as CatalogItemLocalStateEnum.
ExistsInCatalog	Returns True if the Component exists in File Catalog. Rel on return from call to GetFileItems.

Obtaining the ISmComponent Object

You obtain the ISmComponent object as follows:

```
SmComponent = SmComponents.Add( "ComponentName" )
```

ISmMasks

The ISmMasks object represents a collection of Mask strings for getting accompanying files for an SmFile object.

Accompanying files, such as redline files, should be copied to File Catalog during a Check Out or Copy operation when the file they are related to is being copied and they should be deleted from the Catalog when the corresponding file is deleted.

Automatic Delete of Accompanying Files

These accompanying files are also automatically deleted whenever the File Catalog automatically deletes the SmFile file (when SmFile.AllowAutomaticDelete is true). File Catalog recognizes the accompanying files to delete according to the mask list specified for the SmFile object in the File Catalog.

File Formats

The SmMasks items specify file formats in the form:

`$F.$Exxx*` (\$F=filename, \$E = extension)

For example, for file `myfile.sldprt`

`$F.red`

represents `myfile.red`

`$F.$Ered*`

represents `myfile.sldprtredxx` and `myfile.sldprtredyy`

Properties

The ISmMasks object has the following properties:

Property	Description
Item	Returns a member of a collection by position. Returns string. Format: <code>\$F.\$Exxx*</code> (\$F=filename, \$E = extension)

Methods

The ISmMasks object has the following methods:

Method	Description
Add	Adds new Mask string to the collection.
Remove	Removes a Mask string from the collection
IndexOf	Returns the index of the first entry in the collection with a specified Mask

Example

Adds a Mask string to the collection.

```
Dim Masks As SmartFileCatalog.ISmMasks
```

```
Masks.Add "$F.red"
```

ISmFileIdentifiers

The ISmFileIdentifiers object represents a collection of ISmFileIdentifier objects.

This object is a way of specifying a set of SmFile objects as a parameter in methods like GetFileItems.

The interpretation of SmFileIdentifier.KeyValue for all SmFileIdentifier objects in the SmFileIdentifiers collection depends on the value of KeyType:

KeyType	KeyValue
cktFileId	FileIdentifier.KeyValue = SmFile.FileId
cktFullName	FileIdentifier.KeyValue = SmFile.FullName

Properties

The ISmFileIdentifiers object has the following properties:

Property	Description
KeyType	Returns or sets the KeyType of SmFileIdentifiers, one of CatalogKeyTypeEnum, either a file identifier or a full name. Each SmFileIdentifier in the collection must use the specified KeyType.
Item	Returns a member of a collection by position. Returns ISmFileIdentif

Methods

The ISmFileIdentifiers object has the following methods:

Method	Description
Add	Adds a new SmFileIdentifier object to the collection.
Remove	Removes a SmFileIdentifier object from the collection
IndexOf	Returns the index of the first entry in the collection with a specified K Value

Example

Adds an SmFileIdentifier object to the collection.

```
Dim SmFileIdentifier As SmartFileCatalog.ISmFileIdentifier
Dim SmFileIdentifiers As SmartFileCatalog.ISmFileIdentifiers

Set SmFileIdentifiers = SmFileCatalog.NewSmFileIdentifiers

SmFileIdentifiers.KeyType = cktFullName

Set SmFileIdentifier = SmFileIdentifiers.Add("C:\Work\MyFile.bmp")
```


ISmFileIdentifier

The SmFileIdentifier object represents an individual FileIdentifier in the File Catalog.

Properties

The ISmFileIdentifier object has the following properties:

Property	Description
KeyValue	Returns or sets the KeyValue of SmFileIdentifier. The interpretation of KeyValue depends on the value of SmFileIdentifiers.KeyType for the collection to which this object belongs. It can be either FileId or FullName.
ComponentIdentifiers	Retrieves the collection object of ISmComponentIdentifiers for SmComponent objects associated with the SmFile object with SmFileIdentifier. Returns ISmComponentIdentifiers

Obtaining the ISmFileIdentifier Object

You obtain the ISmFileIdentifier object as follows:

Example

The following:

```
Set SmFileIdentifier = SmFileIdentifiers.Add("C:\Work\MyFile.bmp")
```

Note: An SmFileIdentifier object cannot exist without being a member of an SmFileIdentifiers collection. You create an SmFileIdentifier object by adding an identifier string to a collection as in this example.

ISmComponentIdentifiers

The ISmComponentIdentifiers object represents a collection of Component Identifier strings.

The members of an SmComponentIdentifiers collection are the ComponentNames of a set of SmComponent objects. There is no SmComponentIdentifier object in the library.

The SmComponent.ComponentName property is used as the Component Identifier, for example:

```
ComponentIdentifiers.Add (SmComponent.ComponentName)
```

Properties

The ISmComponentIdentifiers object has the following properties:

Property	Description
Item	Returns a member of a collection by position. Returns Component Identifier string.

Methods

The ISmComponentIdentifiers object has the following methods:

Method	Description
Add	Adds new Component Identifier string to the collection.
Remove	Removes a Component Identifier string from the collection
IndexOf	Returns the index of the first entry in the collection with a specified ComponentIdentifier string

Example

Adds a Component Identifier string to the collection.

```
Dim SmFileIdentifier As SmartFileCatalog.ISmFileIdentifier

Dim SmFileIdentifiers As SmartFileCatalog.ISmFileIdentifiers

Dim SmComponentsIdentifiers As SmartFileCatalog.ISmComponentIdentifiers

Set SmFileIdentifiers = SmFileCatalog.NewSmFileIdentifiers

SmFileIdentifiers.KeyType = cktFileId

Set SmFileIdentifier = SmFileIdentifiers.Add("4334-05495454844448")

Set SmComponentsIdentifiers = SmFileIdentifier.ComponentIdentifiers

SmComponentsIdentifiers.Add("Default")
```

ISmFolders

The ISmFolders object represents a collection of ISmFolder objects.

Properties

The ISmFolders object has the following properties:

Property	Description
Item	Returns a member of a collection by position. Returns ISmFolder

Methods

The ISmFolders object has the following methods:

Method	Description
IndexOf	Returns the index of the first SmFolder in the collection with specified path

Example

Get a SmFolder object from the collection.

```
Dim SmFolder As SmartFileCatalog.ISmFolder
```

```
Dim SmFolders As SmartFileCatalog.ISmFolders
```

```
Set SmFolders =
```

```
SmFileCatalog.GetCatalogFolders(SmFileCatalog.CurrentWorkspace)
```

```
Set SmFolder = SmFolders.Item(0)
```

ISmFolder

The SmFolder object represents an individual file Folder in the File Catalog.

A SmFolder object is associated with private files or a shared Workspace and is returned by the GetCatalogFolder method.

Note: For a shared Workspace, an SmFolder object is returned by the GetCatalogFolder method if its physical folder belongs to the directory tree of the Workspace folder – even if there are no SmarTeam files in the folder.

Folders are differentiated by FolderType

FolderType	Description
cftRegular	Regular folder containing private files or belonging to a specific shared Workspace
cftTemporary	A temporary folder is defined for a user but not for a specific shared Workspace. A Temporary folder is used for storing files for viewing. (SmarTeam creates subfolders named by GUIDs under the Temporary Directory for this purpose). The Temporary folder is returned by GetCatalogFolders only for private files.
cftLibrary	A library folder is defined for storing standard parts (not implemented, SmarTeam doesn't pass this folder)
cftDefault	A Default folder is defined for each Workspace. It is normally used as the default working directory.

SmFolder Creation

A registration of the physical folder containing a file is created in the File Catalog when the file object is registered. In the API, the Update method performs the file registration operation.

File Catalog classifies the new folder object as follows: If the path of the folder that contains the file being registered is not in a shared Workspace tree under the root, the folder is classified as a private file folder. If the path of the folder that contains the file being registered is in a shared Workspace tree under the root, then the folder is associated with the shared Workspace.

In the API, a folder object is represented by SmFolder and is uniquely specified by its path, represented by property SmFolder.Path. The collection SmFolderIdentifiers contains path strings that identify the SmFolder objects

The FolderId property does not identify the folder. The FolderId property can be used to determine the presence of SmarTeam files in the folder: If a Catalog folder has SmarTeam files located in it, then the SmFolder.FolderId property is non-zero.

The object diagram of ISmFolder is shown below:

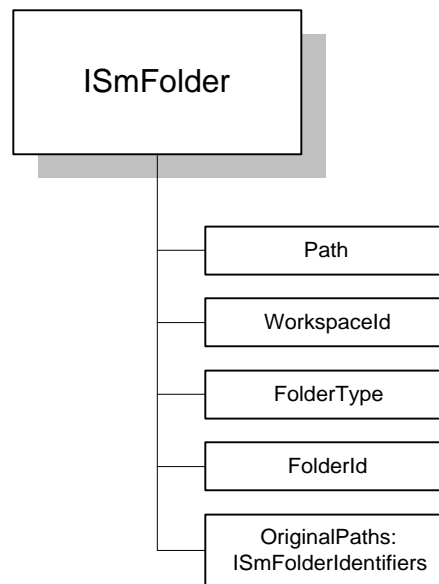


Figure 13-3 ISmFolder Object Diagram

Properties

The ISmFolder object has the following properties:

Property	Description
Path	Returns the path of the folder represented by SmFolder.
WorkspaceId	Returns the Workspace Id of the SmFolder. For private files and shared Workspaces, it represents the Workspace to which the folder belongs. The WorkspaceId property is non-zero if SmFolder is located in a shared workspace directory tree under the root, even if there are SmarTeam files in the folder.
FolderType	Returns the FolderType of SmFolder as CatalogFolderTypeEnum.
FolderId	Returns the FolderIdentifier of SmFolder. A non-zero value of this property for both private files and a shared workspace indicates that the SmFolder.Path contains SmarTeam files. Note: the members of the collection SmFolderIdentifiers are values of SmFolder.Path, and not SmFolder.FolderId.
OriginalPaths	List of previous mappings of this SmFolder Returns ISmFolderIdentifiers

Obtaining the ISmFolder Object

You obtain the ISmFolder object as follows:

Example

The following:

```
Set SmFolders = SmFileCatalog.GetCatalogFolders(SmFileCatalog.  
CurrentWorkspace)
```

```
Set SmFolder = SmFolders.Item(0)
```

ISmFolderIdentifiers

The ISmFolderIdentifiers object represents a collection of path strings that uniquely identify SmFolder objects.

Properties

The ISmFolderIdentifiers object has the following properties:

Property	Description
Item	Returns a member of a collection by position. Returns SmFolder path string.

Methods

The ISmFolderIdentifiers object has the following methods:

Method	Description
Add	Adds new Folder path string to the collection.
Remove	Removes a Folder path string from the collection
IndexOf	Returns the index of the first entry in the collection with a specified Folder path string

Example

Adds a Folder path string to the collection.

```
Dim SmFoldersIdentifiers As SmartFileCatalog.ISmFolderIdentifiers  
  
Set SmFoldersIdentifiers = SmFileCatalog.NewSmFolderIdentifiers  
  
SmFoldersIdentifiers.Add "C:\Work"
```

ISmWorkspaces

The ISmWorkspaces object represents a collection of ISmWorkspace objects.

Properties

The ISmWorkspaces object has the following properties:

Property	Description
Item	Returns a member of a collection by position. Returns ISmWorkspace.

Methods

The ISmWorkspaces object has the following methods:

Method	Description
--------	-------------

IndexOf	Returns the index of the first entry in the collection with a specified id.
---------	---

Example

Get a SmWorkspace object from the collection.

```
Dim SmWorkspace As SmartFileCatalog.ISmWorkspace
```

```
Dim SmWorkspaces As SmartFileCatalog.ISmWorkspaces
```

```
Set SmWorkspaces = SmFileCatalog.RecentWorkspaces
```

```
Set SmWorkspace = SmWorkspaces.Item(0)
```

ISmWorkspace

The SmWorkspace object represents an individual Workspace in the File Catalog.

A Workspace defines a directory tree with one root for the purpose of file storage of shared files used by several users in collaboration and provides a mechanism for security management, all of which provides a basis for Collaborative Design.

In one Workspace you work only with information and files from one database.

An SmWorkspace is characterized by Id, Name and Path. For private files, Name is empty. Path specifies the location of the Workspace. The Workspace name is used for display purposes.

Properties

The ISmWorkspace object has the following properties:

Property	Description
Id	Returns the Id of SmWorkspace
Path	Returns the Path of SmWorkspace
Name	The workspace name is determined by its creator and it used for display purposes.

Obtaining the ISmWorkspace Object

You obtain the ISmWorkspace object as follows:

Example

The following:

```
Set SmWorkspace = SmWorkspaces.Item(0)
```

ISmResultItems

The ISmResultItems object represents a collection of ISmResultItem objects.

In the method GetFileItems(SmFileIdentifiers), the number of elements in the returned parameter SmResultItems is determined according to the value of the KeyType property in the SmFileIdentifiers parameter as follows:

If KeyType is cktFullName, then the file items are uniquely determined by their full name and the number of SmFiles objects in SmResultItems will be the same as the number of files specified in GetFileItems method.

If KeyType is cktFileId, then since file items are not uniquely determined by FileId, there may be more elements in SmResultItems than the number of files specified in the GetFileItems method. This can occur when there exist multiple copies of a file specified by FileId in the GetFileItems method. Each copy is returned as a separate element of ISmResultItems. An exception to this rule occurs when only one of the multiple copies of the file is in the editable state (checked out). In that case, only the editable file copy is returned as an element in SmResultItems and the other copies are ignored.

In a call to the Update method, the KeyType is automatically set to cktFullName.

Properties

The ISmResultItems object has the following properties:

Property	Description
HasErrors	True if any errors have occurred.
KeyType	Returns KeyType of SmResultItems as CatalogKeyTypeEnum. The value of KeyType depends on which method was called: <ul style="list-style-type: none">• For a call to GetFileItems, the value of KeyType that was used in the FileIdentifiers argument is used in SmResultItems.• For a call to the Update method, the value of KeyType = cktFullName is always used in SmResultItems.
Item	Returns a SmResultItem of the collection by position. Returns ISmResultItem.

Methods

The ISmResultItems object has the following methods:

Method	Description
AsSmFiles	Returns the SmFile objects in SmResultItems as a SmFiles collection. Returns ISmFiles.
IndexOf	Returns the index of the first SmResultItem in the collection with a specified file identifier KeyValue.

Example

Get an SmResultItem object from the collection.

```
Dim SmFiles As SmartFileCatalog.ISmFiles
Dim SmFile As SmartFileCatalog.ISmFile
Dim SmComponent As SmartFileCatalog.ISmComponent
Dim SmComponents As SmartFileCatalog.ISmComponents
Dim SmResultItem As SmartFileCatalog.ISmResultItem
Dim SmResultItems As SmartFileCatalog.ISmResultItems
Set SmFile = SmFiles.Add("C:\Work\MyFile.bmp")
SmFile.FileId = "98789021101989849832832"
SmFile.AllowAutomaticDelete = True
SmFile.FileVersion = 0
SmFile.LocalState = cilsEditable
SmFile.Owner = "joe"
SmFile.PrimaryComponentName = "Default"
Set SmComponents = SmFile.Components
Set SmComponent = SmComponents.Add("Default")
Set SmReferences = SmFileCatalog.NewSmReferences
Set SmResultItems = SmFileCatalog.Update(SmFiles, SmReferences, False, True)
If (SmResultItems.HasErrors = False) Then
    Set SmResultItem = SmResultItems.Item(0)
End If
```

ISmResultItem

The SmResultItem object represents an individual ResultItem in the File Catalog.

Properties

The ISmResultItem object has the following properties:

Property	Description
ReturnCode	Return error code for a specific File Item as CatalogReturnCodeEnum.
KeyValue	The value of KeyType used for this KeyValue parameter depends which method was called: <ul style="list-style-type: none">• For a call to GetFileItems, the value of KeyType that was used in the FileIdentifiers argument is used in SmResultItems.• For a call to the Update method, the value of KeyType = cktFullName is always used in SmResultItems.
SmFile	SmFile object. Returns ISmFile.

Obtaining the ISmResultItem Object

You obtain the ISmResultItem object as follows:

```
If (SmResultItems.HasErrors = False) Then  
    Set SmResultItem = SmResultItems.Item(0)  
End If
```

ISmRetrieveFilter

The SmRetrieveFilter object represents a RetrieveFilter in the File Catalog.

Properties

The ISmRetrieveFilter object has the following properties:

Property	Description
RetrieveComponents	Retrieve components according to setting CatalogRetrieveFilterEnum: crfAll or crfSelected. When crfSelected, then only the Components specified by ComponentIdentifier in ISmFileIdentifier are returned with the ISmFile.
RetrieveReferences	True to retrieve SmReference objects.
RetrieveMasks	True to retrieve SmMask objects..
RetrieveAttachedItems	True to retrieve AttachedItems.

Obtaining the ISmRetrieveFilter Object

You obtain the ISmRetrieveFilter object as follows:

```
Dim SmRetrieveFilter As SmartFileCatalog.ISmRetrieveFilter
Set SmRetrieveFilter = SmFileCatalog.NewSmRetrieveFilter
```

Example

The following sets a Retrieve Filter:

```
Dim SmRetrieveFilter As SmartFileCatalog.ISmRetrieveFilter
Set SmRetrieveFilter = SmFileCatalog.NewSmRetrieveFilter
SmRetrieveFilter.RetrieveComponents = crfAll
SmRetrieveFilter.RetrieveMasks = True
SmRetrieveFilter.RetrieveReferences = True
```

Common Tasks

The following sections describe methods and properties that are used to perform common tasks related to a File Catalog and its components.

FileCatalog Task: Update File Item

The following example shows how to register a new SmarTeam file to SmartFileCatalog.

```
Dim SmFileCatalog As SmartFileCatalog.ISmFileCatalog

Dim ClientContextService As SmartClientContextService.SmClientContextService

Set ClientContextService =
SmSession.GetService("SmartClientContextService.SmClientContextService")

Set SmFileCatalog = ClientContextService.ClientContext.FileCatalog

Dim SmFiles As SmartFileCatalog.ISmFiles

Dim SmFile As SmartFileCatalog.ISmFile

Dim SmResultItem As SmartFileCatalog.ISmResultItem

Dim SmResultItems As SmartFileCatalog.ISmResultItems

Dim SmComponent As SmartFileCatalog.ISmComponent

Dim SmComponents As SmartFileCatalog.ISmComponents

Set SmFile = SmFiles.Add("C:\Work\MyFile.bmp")

SmFile.FileId = "98789021101989849832832" 'from Database

SmFile.AllowAutomaticDelete = True

'prevent modification date of file from being updated

SmFile.MarkToSynchronizeWithServer = False

SmFile.FileVersion = 0 'from Database

SmFile.LocalState = cilsEditable

SmFile.Owner = "joe"

SmFile.PrimaryComponentName = "Default"
```

```
Set SmComponents = SmFile.Components

Set SmComponent = SmComponents.Add("Default")

Set SmReferences = SmFileCatalog.NewSmReferences

Set SmResultItems = SmFileCatalog.Update(SmFiles, SmReferences, False, True)

If (SmResultItems.HasErrors = False) Then

    Set SmResultItem = SmResultItems.Item(0)

    ...

End If
```

FileCatalog Task: Get File Item

This example shows how to Get File Item properties registered in SmartFileCatalog.

```
Dim SmFileCatalog As SmartFileCatalog.ISmFileCatalog

Dim ClientContextService As SmartClientContextService.SmClientContextService

Set ClientContextService =
SmSession.GetService("SmartClientContextService.SmClientContextService")

Set SmFileCatalog = ClientContextService.ClientContext.FileCatalog

Dim SmFile As SmartFileCatalog.ISmFile

Dim SmRetrieveFilter As SmartFileCatalog.ISmRetrieveFilter

Dim SmFileIdentifier As SmartFileCatalog.ISmFileIdentifier

Dim SmResultItem As SmartFileCatalog.ISmResultItem

Dim SmResultItems As SmartFileCatalog.ISmResultItems

Dim SmFileIdentifiers As SmartFileCatalog.ISmFileIdentifiers

Set SmFileIdentifier = SmFileIdentifiers.Add("9237678327864690120")

Set SmRetrieveFilter = SmFileCatalog.NewSmRetrieveFilter

SmRetrieveFilter.RetrieveComponents = crfAll

SmRetrieveFilter.RetrieveMasks = True

SmRetrieveFilter.RetrieveReferences = True
```

```
Set SmResultItems = SmFileCatalog.GetFilesItems(SmFileIdentifiers,
SmRetrieveFilter)

If (SmResultItems.HasErrors = False) Then

    Set SmResultItem = SmResultItems.Item(0)

    Set SmFile = SmResultItem.SmFile

    ...

End If
```

FileCatalog Task: Get Catalog Folders

This example shows how to get Catalog folders of Current workspace

```
Dim SmFileCatalog As SmartFileCatalog.ISmFileCatalog

Dim ClientContextService As SmartClientContextService.SmClientContextService

Set ClientContextService =
SmSession.GetService("SmartClientContextService.SmClientContextService")

Set SmFileCatalog = ClientContextService.ClientContext.FileCatalog

Dim SmFolder As SmartFileCatalog.ISmFolder

Dim SmFolders As SmartFileCatalog.ISmFolders

Set SmFolders = SmFileCatalog.GetCatalogFolders(SmFileCatalog.CurrentWorkspace)

For i = 0 To SmFolders.Count - 1

    Set SmFolder = SmFolders.Item(i)

    ...

Next i
```

FileCatalog Task: Get Temporary Files

This example shows how to use GetTemporaryFiles to retrieve the given SmFiles from the user's Temporary folder. Each input SmFile object uniquely specifies a file by FileId and FileVersion.


```
Dim SmFileCatalog As SmartFileCatalog.ISmFileCatalog

Dim ClientContextService As SmartClientContextService.SmClientContextService

Set ClientContextService =
SmSession.GetService("SmartClientContextService.SmClientContextService")

Set SmFileCatalog = ClientContextService.ClientContext.FileCatalog

Dim SmFiles As SmartFileCatalog.ISmFiles

Dim SmFile As SmartFileCatalog.ISmFile

Dim SmResultItem As SmartFileCatalog.ISmResultItem

Dim SmResultItems As SmartFileCatalog.ISmResultItems

Set SmFile = SmFiles.Add("MyTempFile1.bmp") \\ Insert only file name without path

SmFile.FileId = "47298C20-8F20-406C-8962-1C11B552F5A5" \\ From database

SmFile.FileVersion = 0 \\ From database

Set SmFile = SmFiles.Add("MyTempFile2.bmp") \\ Insert only file name

SmFile.FileId = "47298C20-8F20-406C-8962-1C11B552F5A6" \\ From database

SmFile.FileVersion = 0 \\ From database

Set SmResultItems = SmFileCatalog.GetTemporaryFiles(SmFiles)

If (SmResultItems.HasErrors = False) Then

    Set SmResultItem = SmResultItems.Item(0)

    ...

End If
```

FileCatalog Task: Updating a Component name

This example shows how to update a Component name in the File Catalog. This is a complete example, showing how the object information is extracted from the SmarTeam database and updated in the File Catalog. The example uses a script, which is triggered by the updating of the Component name in the SmarTeam object. The BeforeUpdate script gets the new and old component names and the AfterUpdate script calls the routine to update the File Catalog.

```
Const NM_TDM_COMPONENT_NAME = "TDM_COMPONENT_NAME"

Const NM_TDM_FILE_ID = "TDM_FILE_ID"

Const NM_GLB_COMPONENT_NAME_OLD = "SmUDGlobalUpdatedComponentNameOld"

Const NM_GLB_COMPONENT_NAME_NEW = "SmUDGlobalUpdatedComponentNameNew"

Const NM_GLB_FILE_ID = "SmUDGlobalUpdatedComponentFileId"

Const NM_GLB_OBJ_STATE = "SmUDGlobalUpdatedComponentState"
```

BeforeUpdate Script

```
Function BeforeUpdateComponent(ApplHndl As Long,Sstr As String,FirstPar As Long,SecondPar As Long,ThirdPar As Long ) As Integer

    Dim Record1 As Object

    Dim UpdatedObject As ISmObject

    Dim SmartSession As ISmSession

    Dim GlbIdx As Integer

    'Convert ApplHndl to SmSession

    Set SmartSession = SCREXT_ObjectForInterface(ApplHndl)

    'Convert record lists into COM SmRecordList objects

    CONV_RecListToComRecordList FirstPar,Record1

    If Record1.Headers.HeaderExists(NM_TDM_COMPONENT_NAME) And (Not Record1.Headers.HeaderExists(NM_TDM_FILE_ID)) Then
```

```
SmartSession.GlobalData.Value(NM_GLB_COMPONENT_NAME_NEW) =  
Record1.ValueAsString(NM_TDM_COMPONENT_NAME,0)  
  
Set UpdatedObject =  
SmartSession.ObjectStore.RetrieveObject(Record1.ValueAsInteger(NM_CLASS_ID,0),  
Record1.ValueAsInteger(NM_OBJECT_ID,0))  
  
SmartSession.GlobalData.Value(NM_GLB_COMPONENT_NAME_OLD) =  
UpdatedObject.Data.ValueAsString(NM_TDM_COMPONENT_NAME)  
  
SmartSession.GlobalData.Value(NM_GLB_FILE_ID) =  
UpdatedObject.Data.ValueAsString(NM_TDM_FILE_ID)  
  
SmartSession.GlobalData.Value(NM_GLB_OBJ_STATE) =  
UpdatedObject.Data.ValueAsInteger(NM_STATE)  
  
Else  
  
GlbIdx = SmartSession.GlobalData.IndexOf(NM_GLB_COMPONENT_NAME_NEW)  
  
If GlbIdx >= 0 Then  
  
SmartSession.GlobalData.Delete NM_GLB_COMPONENT_NAME_NEW  
  
End If  
  
GlbIdx = SmartSession.GlobalData.IndexOf(NM_GLB_COMPONENT_NAME_OLD)  
  
If GlbIdx >= 0 Then  
  
SmartSession.GlobalData.Delete NM_GLB_COMPONENT_NAME_OLD  
  
End If  
  
GlbIdx = SmartSession.GlobalData.IndexOf(NM_GLB_FILE_ID)  
  
If GlbIdx >= 0 Then  
  
SmartSession.GlobalData.Delete NM_GLB_FILE_ID  
  
End If  
  
GlbIdx = SmartSession.GlobalData.IndexOf(NM_GLB_OBJ_STATE)  
  
If GlbIdx >= 0 Then  
  
SmartSession.GlobalData.Delete NM_GLB_OBJ_STATE  
  
End If  
  
End If  
  
BeforeUpdateComponent=Err_None
```

```
End Function
```

AfterUpdate Script

```
Function AfterUpdateComponent(ApplHndl As Long,Sstr As String,FirstPar As  
Long,SecondPar As Long,ThirdPar As Long ) As Integer
```

```
Dim SmartSession As ISmSession
```

```
Dim GlbIdx As Integer
```

```
AfterUpdateComponent=Err_none
```

```
'Convert ApplHndl to SmSession
```

```
Set SmartSession = SCREXT_ObjectForInterface(ApplHndl)
```

```
GlbIdx = SmartSession.GlobalData.IndexOf(NM_GLB_COMPONENT_NAME_NEW)
```

```
If GlbIdx >= 0 Then
```

```
    AfterUpdateComponent=UpdateSmartCatalog( SmartSession,  
SmartSession.GlobalData.Value(NM_GLB_COMPONENT_NAME_OLD),  
SmartSession.GlobalData.Value(NM_GLB_COMPONENT_NAME_NEW),  
SmartSession.GlobalData.Value(NM_GLB_FILE_ID),  
SmartSession.GlobalData.Value(NM_GLB_OBJ_STATE))
```

```
End If
```

```
End Function
```

Subroutine for Updating File Catalog

```
Function UpdateSmartCatalog( SmartSession As Object, _  
    OldComponentName As String, _  
    NewComponentName As String, _  
    FileId As String, _  
    ObjectState As Integer) As Integer  
    ' Obtaining the ISmFileCatalog Objects  
  
    Dim SmFileCatalog As Object  
    Dim ClientContextService As Object  
  
    Set ClientContextService =  
    SmartSession.GetService("SmartClientContextService.SmClientContextService")  
  
    Set SmFileCatalog = ClientContextService.ClientContext.FileCatalog  
    Dim Index As Integer  
  
    ' Create SmFileIdentifiers for the object to retrieve  
    Set SmFileIdentifiers = SmFileCatalog.NewSmFileIdentifiers  
    SmFileIdentifiers.KeyType = 1 ' CatalogKeyTypeEnum.cktFileId  
    FileIdStr = FileId  
    SmFileIdentifiers.Add FileIdStr  
  
    ' Create SmRetrieveFilter to retrieve all components  
    Set SmRetrieveFilter = SmFileCatalog.NewSmRetrieveFilter  
    SmRetrieveFilter.RetrieveComponents = 0 ' CatalogRetrieveFilterEnum.crfAll  
  
    ' Execute GetFileItems operation  
  
    Set SmResultItems = SmFileCatalog.GetFileItems(SmFileIdentifiers,  
    SmRetrieveFilter)  
  
    If SmResultItems.HasErrors = False Then
```

```
' Get the first item in the SmResultItems, you might get more the one object
if the file is checked in

' and there in more then one copy file in your disk

For i=0 To SmResultItems.Count - 1

    Set SmResultItem = SmResultItems.Item(i)

    If SmResultItem Is Not Nothing Then

        Set SmFile = SmResultItem.SmFile

        If SmFile.LocalState = 1 Then ' cilsEditable

            Set SmComponents = SmFile.Components

            If SmComponents Is Not Nothing Then

                oldComponentNameStr = OldComponentName

                Index = SmComponents.IndexOf(oldComponentNameStr)

                If Index <> -1 Then

                    ' Remove the old component

                    intIndex = Index

                    SmComponents.Remove intIndex

                    ' Add the new component

                    newComponentNameStr = NewComponentName

                    Set SmComponent = SmComponents.Add(newComponentNameStr)

                    If ObjectState = 0 Or ObjectState = 2 Then ' new or checked out

                        SmComponent.LocalState = 1 ' cilsEditable

                    ElseIf ObjectState = 1 Or ObjectState = 3 Then ' checked in or
released

                        SmComponent.LocalState = 3 ' cilsNotEditable

                    Else

                        SmComponent.LocalState = 0 ' cilsUndefined

                    End If

                    ' Create new SmFiles object
```

```
Set newSmFiles = SmFileCatalog.NewSmFiles

Set newSmFile = newSmFiles.Add(SmFile.FullName)

newSmFile.FileId = SmFile.FileId

Set newSmComponents = newSmFile.Components

For j=0 To SmComponents.Count - 1

    newSmComponents.Add SmComponents.Item(j).Name

    newSmComponents.Item(j).LocalState =
SmComponents.Item(j).LocalState

Next j

Set newSmMasks = newSmFile.Masks

For j=0 To SmFile.Masks.Count - 1

    newSmMasks.Add SmFile.Masks.Item(j)

Next j

Set SmReferences = SmFileCatalog.NewSmReferences

' Update the object with the new component

Dim boolValue

boolValue = False

Set SmResultItems2 = SmFileCatalog.Update(newSmFiles, SmReferences,
boolValue, boolValue)

If SmResultItems2.HasErrors = False Then

    UpdateSmartCatalog = Err_None

End If

End If

End If

End If

Next i

End If

End Function
```

14. SmartRecordList Library

General Description

The SmartRecordList library comprises objects that enable the following functionality:

- Allows a client to work with record list data objects that are similar to those in the SmarTeam API

Dependencies

The **SmarTeam** File Catalog library has the following dependencies:

- SmarTeam Record List library
- SmarTeam Client Services.

Overview of Record List Objects

The SmartRecordList Library provides two record list types: `ImmutableRecordList` and `IRecordList`.

The purpose of these Record List objects is to allow the client to work with record list data objects that are similar to those in the SmarTeam API (see Chapter 4, [SmarTeam Record List Library](#)).

For example, a client can request that SmarTeam return information to him in a Record List.

The `ImmutableRecordList` is provided for working with and modifying the data in the record list. The `IRecordList` is provided when you want to read data while preserving it.

The two Record List objects are similar in structure. The main difference is that you can create read-only objects from the read-write objects but not vice versa.

IMutableRecordList

The following figure shows the object diagram for the MutableRecordList Object.

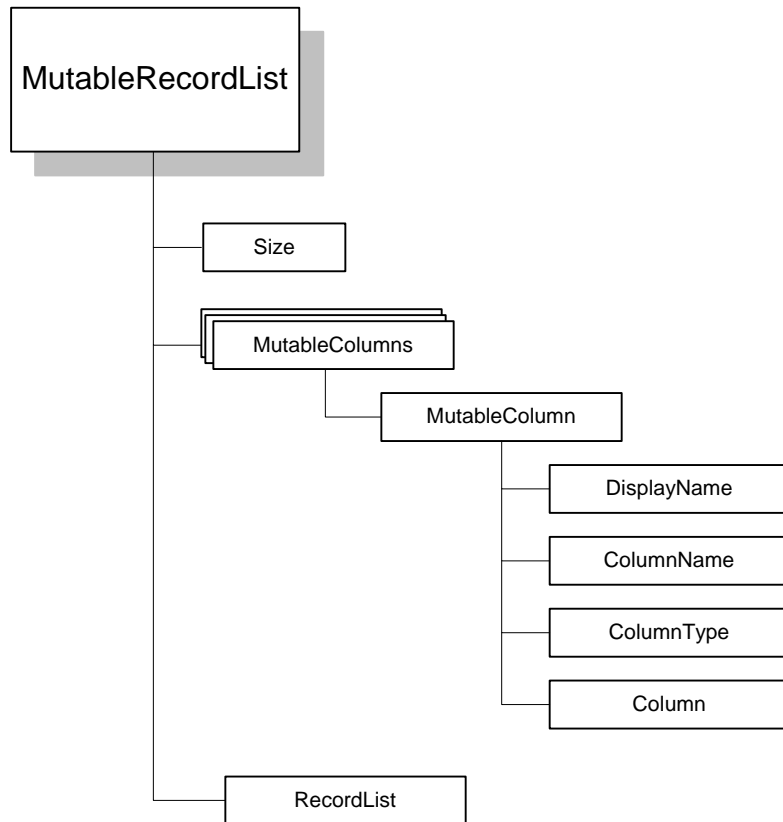


Figure 4 MutableRecordList Object Diagram

Properties

The MutableRecordList object has the following properties:

Property	Description
Columns	Returns the set of columns associated with the MutableRecordList, as IMutableColumns.
RecordList	Returns a read-only copy of this MutableRecordList, as IRecordList.
Size	Number of MutableRecord objects in this MutableRecordList.

Methods

The MutableRecordList object has the following methods:

Method	Description
AddRecord	Add a MutableRecord to the MutableRecordList. Returns IMutableRecord.
AddRecordListChangeListener	Adds specified RecordListChangeListener to this MutableRecordList.
AddRecordListValueChangeListener	Adds a RecordListValueChangeListener to this MutableRecordList.
Clear	Deletes all headers and nodes in this MutableRecordList object.
CompareTo	Compares this MutableRecordList to a specified MutableRecordList. If all nodes are equal it returns 0, otherwise it returns a non-zero value.
Create	Creates MutableRecordList with the headers of a specified IRecordList. All nodes are set to nil.
EnableMutableRecordListEvents	Enable the MutableRecordListEvents for this MutableRecordList.
GetFilteredIterator	Returns a filtered RecordListIterator. The filtered iterator retrieves records from the RecordList that satisfy the condition specified by ICondition.
GetIterator	Returns a simple RecordListIterator. The RecordListIterator can be synchronized to the MutableRecord List.
GetSortedIterator	Returns a sorted RecordListIterator. The sorted iterator retrieves records from the RecordList sorted according to the Comparator object.
RemoveRecordListChangeListener	Removes a RecordListChangeListener from this MutableRecordList.
RemoveRecordListValueChangeListener	Removes a RecordListValueChangeListener from this MutableRecordList.

IMutableColumns

Properties

The MutableColumns object has the following properties:

Method	Description
Columns	Retrieves a read-only object of MutableColumns. Returns IColumns.
MutableColumn	Gets a MutableColumn by index. Returns IMutableColumn.
Size	Returns the number of MutableColumn objects in the collection.

Methods

The IMutableColumns object has the following methods:

Method	Description
AddColumn	Adds a MutableColumn, specified by header name and data type, to the collection.
AddColumnsChangeListener	Add a ColumnsChangeListener to the collection.
AddPropertyChangeListener	Adds a PropertyChangeListener to the collection.
Clear	Clears headers and nodes of all MutableColumn objects in the collection.
CompareTo	Compares the MutableColumn objects in the collection to the MutableColumn objects in the argument collection. Returns 0 if all corresponding MutableColumn objects are equal, otherwise returns 1. Two MutableColumns are equal if the ColumnType, DisplayName and ColumnName are equal.
EnableMutableColumnsEvents	Enables or disables the events fired by the MutableColumns object.
IndexOf	Returns the index of a specified MutableColumn in the collection.
RemoveColumn	Removes a MutableColumn specified by position.
RemoveColumnsChangeListener	Removes a ColumnsChangeListener from the collection.
RemovePropertyChangeListener	Removes a PropertyChangeListener from the collection.

IMutableColumn

Properties

The MutableColumn object has the following properties:

Property	Description
Column	Retrieves a read-only object of MutableColumn. Returns IColumn (Ref).
ColumnName	Gets the header name of this MutableColumn.
ColumnType	Gets the data type of this MutableColumn.
DisplayName	Gets the display name for this MutableColumn.

Methods

The MutableColumn object has the following methods:

Method	Description
AddPropertyChangeListener	Adds a PropertyChangeListener to the MutableColumn.
CompareTo	Compare this MutableColumn to a specified MutableColumn. Returns 0 if the ColumnType, DisplayName and ColumnName are equal, otherwise returns 1.
Create	Creates a MutableColumn, according to specified header name and data type.
RemovePropertyChangeListener	Removes a PropertyChangeListener from the MutableColumn.

IMutableRecord

Properties

The MutableRecord object has the following properties:

Property	Description
Columns	Returns the set of MutableColumns associated with the MutableRecord, as IMutableColumns.
Record	Returns a read-only copy of this MutableRecord as IRecord.

Methods

The MutableRecord object has the following methods:

Method	Description
AddRecordChangeListener	Adds a RecordChangeListener to this MutableRecord.
Clear	Clears the nodes of this MutableRecord.
CompareTo	Compares this MutableRecord to a specified MutableRecord. Returns 0 if every node value is equal, otherwise returns a non-zero value.
EnableMutableRecordEvents	Enable the MutableRecordEvents for this MutableRecord.
GetValue(index)	Returns a node value according to node index.
GetValueAs[Boolean, Byte, Double, Float, Long, Short, String] (index)	Returns a node value according to node index, which casts back to simple values instead of objects (int instead of an Integer object).
GetValueByName(ColumnName)	Returns a node value according to header name.
RemoveRecordChangeListener	Removes a RecordChangeListener from this MutableRecord.
SetValue(index, newVal)	Sets the value of a specific location in the MutableRecord.
SetValueAs[Boolean, Byte, Double, Float, Long, Short, String] (index, newVal)	Sets a value with casting of simple types to objects (No need to cast int to Integer object)
SetValueByName(ColumnName, newVal)	Sets a value in the MutableRecord according to header name.

IRecordList

The following figure shows the object diagram for the RecordList Object.

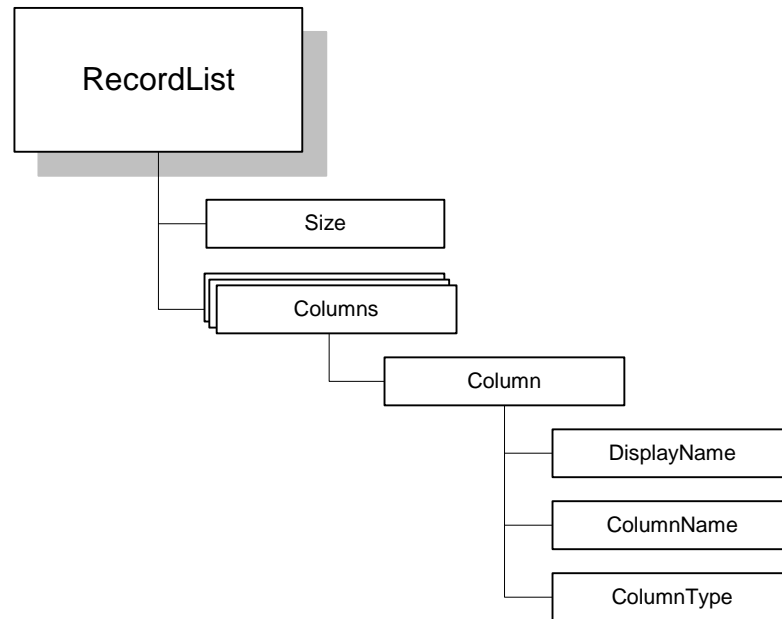


Figure 5 RecordList Object Diagram

IRecordList and its associated interfaces are a read-only version of the IMutableRecordList object. They are used in the same way, with the following exceptions:

All getValue methods work as in the IMutableRecord object; the setValue methods do not. You cannot set the values of an IRecord object; you can only read them.

You cannot create an IMutableRecordList, IMutableRecord or IMutableColumn from an IRecordList, IRecord or IColumn object.

Properties

The RecordList object has the following properties:

Property	Description
Columns	Returns the set of Column objects associated with this RecordList as IColumns.
Size	Returns the number of records in this RecordList.

Methods

The RecordList object has the following methods:

Method	Description
AddRecord	Add an Record to this RecordList. Returns IRecord.
AddRecordListChangeListener	Adds specified RecordListChangeListener to this RecordList.
AddRecordListValueChangeListener	Adds a RecordListValueChangeListener to this RecordList.
CompareTo	Compares this RecordList to a specified RecordList. Returns 0 if Records are equal, otherwise returns a non-zero value. Two Records are equal if all nodes are equal.
EnableRecordListEvents	Enable the RecordListEvents for this RecordList.
GetFilteredIterator	Returns a filtered IRecordListIterator. The filtered iterator retrieves records from the RecordList that satisfy the condition specified by ICondition.
GetIterator	Returns a simple RecordListIterator. The RecordListIterator is synchronized to the Record List: when a record is removed from the RecordListIterator, the corresponding record is removed from the RecordList (?).
GetSortedIterator	Returns a sorted IRecordListIterator. The sorted iterator retrieves records from the RecordList sorted according to the Comparator object.
RemoveRecordListChangeListener	Removes a RecordListChangeListener from this RecordList.
RemoveRecordListValueChangeListener	Removes a RecordListValueChangeListener from this RecordList.

IColumns

Properties

The Columns object has the following properties:

Property	Description
Column	Gets a Column by index. Returns IColumn.
Size	Returns the number of Column objects in the collection.

Methods

The Columns object has the following methods:

Method	Description
AddColumnsChangeListener	Add a ColumnsChangeListener to the collection.
AddPropertyChangeListener	Adds a PropertyChangeListener to the collection.
CompareTo	Compares the Column objects in the collection to the Column objects in the argument collection. Returns 0 if all corresponding Column objects are equal, otherwise returns 1. Two Columns are equal if the ColumnType, DisplayName, and ColumnName are equal.
EnableColumnsEvents	Enables or disables the events that are fired by the Columns object.
IndexOf	Returns the index of a member Column specified by ColumnName.
RemoveColumnsChangeListener	Removes a ColumnsChangeListener from the collection.
RemovePropertyChangeListener	Removes a PropertyChangeListener from the collection.

IColumn

Properties

The Column object has the following properties:

Property	Description
ColumnName	Gets the header name of this Column.
ColumnType	Gets the data type in this Column.
DisplayName	Gets the display name of this Column.

Methods

The Column object has the following methods:

Method	Description
AddPropertyChangeListener	Adds a PropertyChangeListener to the Column.
CompareTo	Compare this Column to a specified Column. Returns 0 if the ColumnType, DisplayName and ColumnName are equal, otherwise returns 1.
RemovePropertyChangeListener	Removes a PropertyChangeListener from the Column.

IRecord

Properties

The Record object has the following properties:

Property	Description
Columns	Returns the set of Column objects associated with the Record, as IColumns.

Methods

The Record object has the following methods:

Method	Description
AddRecordChangeListener	Adds a RecordChangeListener to this Record.
CompareTo	Compares this Record to a specified Record. Returns 0 if every node value is equal, otherwise returns a non-zero value.
EnableRecordEvents	Enables the RecordEvents for this Record.

GetValue(index)	Returns a node value according to node index.
GetValueAs[Boolean, Byte, Double, Float, Int, Long, Short, String] (index)	Returns a node value according to node index, which comes back to simple values instead of objects (int instead of Integer object).
GetValueByName(ColumnName)	Returns a node value according to column name.
RemoveRecordChangeListener	Removes a RecordChangeListener from this Record.

IRecordListIterator

Methods

The IRecordListIterator object has the following methods:

Method	Description
Find	Moves to the next position according to the given ICondition, Else moves to the end of the list.
FindNext	Moves to the next position according to the ICondition specified in the last Find method, Else moves to the end of the list.
HasNext	Returns true if the iteration has more elements.
IsSynchronized	Returns true if the Iterator is synchronized to the IRecordList.
Next	Returns the next element in the iteration.
Remove	Removes from the underlying collection the last element returned by the iterator.

IRecordListUtils

Methods

The RecordListUtils object has the following methods:

Method	Description
GetGroupCount	Returns the number of groups in the specified RecordList.
GetGroupName	Returns the Group Name of the Column of the specified RecordList at the specified index.
GetMutableGroupCount	Returns the number of groups in the specified MutableRecordList.
GetMutableGroupName	Returns the Group Name of the Column of the specified MutableRecordList at the specified index.

GetMutableRecordListByGroup	Returns a sub MutableRecordList that contains only the Column objects of the specified group. Returns IMutableRecordList.
GetRecordListByGroup	Returns a sub RecordList that contains only the Column objects of the specified group. Returns IRecordList.
SaveToFile	Save To File (not implemented).

IRecordsFactory

Methods

The RecordsFactory object has the following methods:

Method	Description
CreateMutableRecord	Creates an empty MutableRecord
CreateMutableRecordFromColumns	Creates a MutableRecord with same Column objects the specified MutableColumns

Events

The SmartRecordList Library has four event types:

- **ColumnsChangeEvent** - Fires on change of Columns object
- **RecordChangeEvent** - Fires on change of Record object
- **RecordListChangeEvent** - Fires on change of RecordList object, such as adding a record to the RecordList
- **RecordListValueChangeEvent**- Fires on change of a record in a RecordList

All events have a `stop()` method that can be called by the listener to prevent the action from occurring. The event source checks the event stop flag before executing the action.

IColumnsChangeEvent

Methods

The ColumnsChangeEvent object has the following methods:

Method	Description
GetColumnChanged	Gets Column that was changed.
GetSource	Gets the object that threw the event.
Initialize	Sets the event.
IsStopped	Checks if the event is stopped.
Stop	Disables the event procedure.

IColumnsChangeListener

Methods

The ColumnsChangeListener object has the following methods:

Method	Description
ColumnBeforeAdd	Called before adding a column.
ColumnAfterAdd	Called after adding a column.
ColumnBeforeRemove	Called before removing a column.
ColumnAfterRemove	Called after removing a column.

IRecordChangeEvent

Methods

The RecordChangeEvent object has the following methods:

Method	Description
GetIndex	Gets index that was changed.
GetSource	Gets the object that threw the event.
Initialize	Sets the event.
IsStopped	Checks if the event is stopped.
Stop	Disables the event procedure.

IRecordChangeListener

Methods

The RecordChangeListener object has the following methods:

Method	Description
ValueBeforeChange	Called before changing a record value.
ValueAfterChange	Called after changing a record value.

IRecordListChangeEvent

Methods

The RecordListChangeEvent object has the following methods:

Method	Description
GetRecordChanged	Gets Record that was changed.
GetSource	Gets the object that threw the event.
Initialize	Sets the event.
IsStopped	Checks if the event is stopped.
Stop	Disables the event procedure.

IRecordListChangeListener

Methods

The RecordListChangeListener object has the following methods:

Method	Description
RecordBeforeAdd	Called before adding a Record
RecordAfterAdd	Called after adding a Record
RecordBeforeRemove	Called before removing a Record
RecordAfterRemove	Called after removing a Record

IRecordListValueChangeEvent

Methods

The RecordListValueChangeEvent object has the following methods:

Method	Description
GetIndex	Gets the index that was changed.
GetRecordChanged	Gets Record that was changed.
GetSource	Gets the object that threw the event.
Initialize	Sets the event.
IsStopped	Checks if the event is stopped.
Stop	Disables the event procedure.

IRecordListValueChangeListener

Methods

The RecordListValueChangeListener object has the following methods:

Method	Description
ValueBeforeChange	Called before changing a specified Record value in the RecordList.
ValueAfterChange	Called after changing a specified Record value in the RecordList.

Common Tasks

The following sections describe methods and properties that are used to perform common tasks in client applications.

IMutableRecordList: Creating and setting values

```
Dim RecList As MutableRecordList

Dim MutableColumns As IMutableColumns

Dim Column As IMutableColumn

// Create new record list object
Set RecList = New MutableRecordList

// Set the columns for the new record list
Set MutableColumns = RecList.Columns

Set Column = MutableColumns.AddColumn("ID", rldtShort)
Column.DisplayName = "worker ID"

// Add record

Dim Record As IMutableRecord

Set Record = recList.AddRecord

Record.SetValueByName "ID", 101

...
```

IMutableRecordList: Usage of ICondition interface and filtered iterators

```
Implements ICondition
```

```
Private Function ICondition_Evaluate(ByVal pRecVal As
SmartRecordList.IMutableRecord) As Boolean

End Function

Dim Cond As New Class1

Dim RecList As MutableRecordList
Dim Iterator As IRecordListIterator
Set Iterator = RecList.GetFilteredIterator(Cond, True)

Do While (Iterator.HasNext())
    Set TmpRecord = Iterator.Next()
    Dim i As Integer
    For i = 0 To TmpRecord.Columns.Size
        Dim tmp As Integer
        tmp = TmpRecord.GetValueAsShort(i)
    Next i
Loop
```

ImmutableRecordList: Adding a RecordListChangeListener

The following example adds a RecordListChangeListener to a MutableRecordList.

```
Implements IRecordListChangeListener
```

```
Private Sub IRecordListChangeListener_RecordAfterAdd(ByVal pVal As  
SmartRecordList.IRecordListChangeEvent)
```

```
End Sub
```

```
Private Sub IRecordListChangeListener_RecordAfterRemove(ByVal pVal As  
SmartRecordList.IRecordListChangeEvent)
```

```
End Sub
```

```
Private Sub IRecordListChangeListener_RecordBeforeAdd(ByVal pVal As  
SmartRecordList.IRecordListChangeEvent)
```

```
End Sub
```

```
Private Sub IRecordListChangeListener_RecordBeforeRemove(ByVal pVal As  
SmartRecordList.IRecordListChangeEvent)
```

```
End Sub
```

```
Dim RecList As MutableRecordList
```

```
Dim Listener As New Class2
```

```
Set RecList = New MutableRecordList
```

```
RecList.AddRecordListChangeListener Listener
```

Appendix A - Tips for Writing Scripts

A script is program code that can be executed in the **SmarTeam** system, usually in response to an event.

With scripts, the user can customize and enhance the **SmarTeam** family of products.

Scripts can be attached to a variety of **SmarTeam** events and executed in a number of ways. Scripts can be attached either to system operations or to specified **SmarTeam** events.

Scripts should be written in Basic with the Smart BasicScript editor. The scripts should be located within the script directory specified in the System Configuration Editor under “Miscellaneous Configuration/Directory Structure”, key “ScriptDirectory”. The script directory can be, for example, *c:\Program Files\SmarTeam\Script*. A script file name should not exceed 50 characters.

Script argument structures can differ from one event to another. The script programmer should be familiar with the script interface at the particular event hook before beginning to write code.

In **SmarTeam** there are two types of the interfaces between main application and embedded script:

- Procedural API interface
- COM interface; all new script hooks, mainly event hooks of WorkFlow.

The following description shows how to switch to COM API within API procedural interface and how to use procedural API within script invoked through COM interface.

How to switch to COM API within procedural script interface

In order to work with SmarTeam object model from within the script that was invoked through a procedural API interface, you need to get the current Session object and convert procedural record lists to COM representation. In some script hooks, second or third record lists are used for transferring data to main applications. You have two options to implement that:

- Use `CONV_RecListToComRecordList` to convert procedural record list, obtained as the parameter of the script, to COM representation, fill the resulting list with values and copy it back to procedural representation using function `ComRecListToRecordList`. The example below show how to implement this approach.
- Use procedural record list API functions in order to fill values directly to the procedural record list, received as the parameter of the script.

Example

```

*****

'      Template for writing scripts

' ApplHndl  - application handle

' FirstPar  - record list usually contains attributes of the selected
object(s)

' SecondPar - record list usually used for transfer of service data between

'
script and application

' ThirdPar  - record list usually for transfer of data from the script to the
application

*****

Declare Sub CONV_RecListToComRecordList Lib "SmTdm32" (ByVal RecList As
Long,ByRef COMRecList As ISmRecordList)

Declare Sub CONV_ComRecListToRecordList Lib "SmTdm32" (ByVal ComRecList As
ISmRecordList,ByRef RecList As Long)

Function TemplateFunc (ApplHndl As Long,Sstr As String,FirstPar As
Long,SecondPar As Long,ThirdPar As Long ) As Integer

    Dim SmSession      As ISmSession

    Dim COMFirstList  As ISmRecordList

    Dim COMSecondList As ISmRecordList

    Dim COMThirdList  As ISmRecordList

    ' Get Session from Application handle

    Set SmSession = SCREXT_ObjectForInterface(ApplHndl)

```

```
' Convert Record lists to COM representation

CONV_RecListToComRecordList FirstPar,COMFirstList

CONV_RecListToComRecordList SecondPar,COMSecondList

CONV_RecListToComRecordList ThirdPar,COMThirdList

...

<Body of the script>

' Copy values from COM Record List to procedural record lists that must be
returned to main application

CONV_ComRecListToRecordList COMThirdList,ThirdPar

End Function
```

How to use procedural API functions in COM interface script

In order to work with functions of procedural API you need to have Application handle. Use function SCREXT_ObjectForInterface to get ApplHndl from the Session object as shown in the following example.

```
Function AfterSendReject(FlowSession As Object, FlowProcess As Object, Node As
Object, Response As Object) As Integer

Dim ApplHndl As Long

ApplHndl = SCREXT_ObjectForInterface(FlowSession.FlowStore.Session)

AfterSendReject = ERR_NONE

End Function
```

Programming Constants

The **SmartTeam** API provides the SmartConstants object, which allows developers writing in scripting languages, such as VBScript and JScript, to use constants (enumerations) without using their actual numeric value, or having to redeclare them in their own code. SmartConstants provides easy access to these constants. The SmartConstants constants are described in the reference guide, for each type library.

Following are some VBScript code samples that demonstrate the use of SmartConstants.

Example

```
Set Constants = CreateObject("SmartConstants.SmConstants")  
  
RecordList.AddHeader "Header1", 255, Constants.SmRecList.sdtChar
```

Once you have established a reference to the SmConstants object, the syntax is `Constants.library-name.constant-name`. You can also use the same instance of the object to access several libraries.

Example

```
Set Constants = CreateObject("SmartConstants.SmConstants")  
  
RecordList.AddHeader "Header1", 255, Constants.SmRecList.sdtChar  
  
Set B = Session.ObjectStore.DefaultBehavior.Clone  
  
B.ConfirmOperations = Constants.SmApplic.coYesToAll
```

You can also keep a reference to the library specific instance:

Example

```
Set Constants = CreateObject("SmartConstants.SmConstants")  
  
Set C2 = Constants.SmRecList  
  
RecordList.AddHeader "Header1", 255, C2.sdtChar
```

To make this a bit more convenient, we've added library specific shortcuts from some of the main objects. So `RecordList.Constants` is a short cut to the constants object for the `SmRecList` TLB, and `Session.Constants` / `Engine.Constants` are shortcuts to the constants object for the `SmApplic` TLB. So the above code can also be written without creating the `Constants` object, in the following manner:

Example

```
RecordList.AddHeader "Header1", 255, RecordList.Constants.sdtChar  
  
Set B = Session.ObjectStore.DefaultBehavior.Clone  
  
B.ConfirmOperations = Session.Constants.coYesToAll
```

Currently, the objects that support these shortcuts are `RecordList`, `Record`, `Engine` and `Session`, with more to follow.

A few more things to note:

- Delphi, Visual Basic and C++ developers do not need to use these objects. It is much more efficient to use the constants defined in the `_TLB.pas` and `.H` files.
- The `SmConstants` object loads the type-library information dynamically, so there is no need to recompile it when new libraries are added or new values are added to an existing library.

SmartScript Editor Workarounds

The following workarounds are used in the following conditions:

- Using the SmartScript Editor with the `SmarfFileCatalog` library, objects such as, `ISmFolder`, and `ISmFileIdentifier`, error "type mismatch" is displayed

To avoid this error, use the following workaround:

- The variables for the `SmarfFileCatalog` library Objects should not be defined as an exact data type but as a `Variant` type.
- In the SmartScript Editor while using properties with arguments, such as, `Property Value (HeaderName,RecordIndex)`. The following error message is displayed, "...Too many parameters encountered".

To avoid this error use either of the following workarounds:

- When using a `SmRecordList/SmRecord` object(s), their variables need to be defined as generic OLE Object:

```
Dim SmRecList as Object
```

—

SmRecList.Value("TDM_ID",0) = "test"

- When working with the ISmObject object, use either of the following workarounds:

- a. Define the variable as its exact data type ISmObject, but call for the Property Value via another Property Data:

```
Dim FolderObject as ISmObject
```

```
FolderObject.Data.Value("TDM_ID")="1111"
```

- b. Define variable as OLE Object:

```
Dim FolderObject as Object
```

```
FolderObject.Value("CN_ID")="1111".
```

ISM Objects Workaround

The following are specific Conditions that are disregarded in ISM Objects:

- ISMObject.RetrieveParentsAndLinks
- ISMObject.RetrieveParentsAndLinksEx
- ISMObject.RetrieveChildrenAndLinks
- ISMObject.RetrieveChildrenAndLinksEx
- ISMObject.RetrieveRelationsAndLinks
- ISMObject.RetrieveRelationsAndLinksEx
- ISMObject.RetrieveParents
- ISMObject.RetrieveParentsEx
- ISMObject.RetrieveChildren
- ISMObject.RetrieveChildrenEx
- ISMObject.RetrieveRelations
- ISMObject.RetrieveRelationsEx

CAD Integration

Integration Name	Integration Class	Integration_Behavior
CATIA	CATIA_PART	TDM_CATIA_PART
"	"	"

Appendix B - SmarTeam Add-In Services

The following is a list of **SmarTeam** add-in services libraries:

SmarTeam Library Name	ProgId
GUI Services	SmGUISrv.SmCommonGUI
Utilities	SmUtil.SmSessionUtil
SmartFlow	SmartFlow.SmFlowStore
SmartMessages	SmartMessages.SmMessageStore
Integration Tools	SmIntegrationTool.SmIntegrationStore
CAD Interface	SmCad.SmCADInterface

Appendix C - Writing Server Applications

This appendix describes how to use the **SmarTeam** COM API to create server-type applications.

Requirements

The requirements of a server-type application are normally as follows:

1. The application should be able to handle multiple concurrent requests from several clients
2. The application should not display a user interface while running.

Guidelines

The following guidelines help you to use the **SmarTeam** API to create server applications that meet the above requirements and that function in a correct and efficient manner:

Guideline 1: Use `SmApplic.SmFreeThreadedEngine` instead of `SmApplic.SmEngine`.

`SmFreeThreadedEngine` is almost identical to `SmEngine`. The main difference between the two objects is the threading model: `SmEngine` is marked as “Both”, while `SmFreeThreadedEngine` is marked as “Free”. The “Both” threading model used by `SmEngine` can result in significantly slower performance, and even in deadlocks. These problems do not occur when using `SmFreeThreadedEngine`.

You can use `SmFreeThreadedEngine` exclusively. `SmEngine` and its old threading model are retained only for backward compatibility and continue to work well for single-threaded applications. However, you must use `SmFreeThreadedEngine` in multi-threaded applications.

Guideline 2: Create `SmSession` objects explicitly; do not use `SmEngine.CreateSession`

Using the old `SmEngine.CreateSession` to create a `SmSession` object causes the object to be created in the Engine apartment. This is not the correct behavior in a multi-threaded application. Instead, you should create the `SmSession` object explicitly, using the mechanism appropriate for your environment: `CreateObject` in Visual Basic, `Server.CreateObject` in ASP applications, and `CoCreateInstance` in C, and so on.

After creating the session, `SmSession.Init` must be called to associate the new session with the `SmEngine` object.

Code that that was written using `CreateSession` will continue to work. New code should use the new style, and multi-threaded applications must use the new style.

Guideline 3: Set `SmEngine.ServerMode` to `True`

Setting the property `SmEngine.ServerMode` to `True` right after the initialization stage (`SmEngine.Init`) prevents the **SmarTeam** API from displaying message dialog boxes on the screen. Displaying a dialog box has the effect of suspending a server application since no operator is available to respond to it.

Guideline 4: Call `SmSession.Close` when the session is no longer required

Call the `SmSession.Close` method as soon as the session is not required to release the server resources for other sessions. Failing to call this method can result in errors during the `SmEngine.Terminate` method.

Guideline 5: Follow the rules of creating Win32 multi-threaded COM applications

Check the MSDN and other Microsoft documentation for a description of the rules that apply when creating Win32 multi-threaded applications.

Appendix D - SmarTeam Integration and Integration Link Behaviors

SmarTeam Integration Behavior and Integration Link Behavior refer to a common functionality that can be imposed on an object.

SmarTeam Integration_Behaviors

The following is a list of SmarTeam Integration_Behaviors:

Integration	Integration Behavior	Description
CATIA	TDM_UG_ASSEMBLY	UG Assembly
CATIA	TDM_CATIA_DESIGN_TABLE	CATIA Design Table
CATIA	TDM_CATIA_ANAL	CATIA Analysis
CATIA	TDM_CATIA_CATALOG	CATIA Catalog
CATIA	TDM_CATIA_DRAWING	CATIA Drawing
CATIA	TDM_CATIA_MATERIAL	CATIA Material
CATIA	TDM_CATIA_MODEL	CATIA Model
CATIA	TDM_CATIA_PART	CATIA Part
CATIA	TDM_CATIA_PRODUCT	CATIA Product
CATIA	TDM_CATIA_PROCESS	CATIA Process
CATIA	TDM_CATIA_SHEET	CATIA Sheet
CATIA	TDM_CATIA_INTCOM	CATIA Internal Component
CATIA	TDM_CATIA_RESULTDOC	CATIA Result Document
CATIA	TDM_CATIA_DOCUMENT	CATIA Document
CATIA	TDM_CATIA_REPRESENT	CATIA Representation
CATIA	TDM_CATIA_ANALCOMP	CATIA Analysis Computations
CATIA	TDM_CATIA_ANALINPUT	CATIA Analysis Input
CATIA	TDM_NC_DOCUMENT	NC
CATIA	TDM_CAD_DOCUMENT	CAD Document
CATIA	TDM_CATIA_ANALRESULT	CATIA Analysis Results
CATIA	TDM_CATIA_CADAM	CATIA CADAM
CATIA	TDM_CATIA_PROCESS_LIB	CATIA Process Library
CATIA	TDM_CATIA_SYSTEM	CATIA System
CATIA	TDM_CATIA_FEATURE_DIC	CATIA Feature Dictionary
CATIA	TDM_UG_PART	UG Part
Microsoft Excel	TDM_EXCEL_DOCUMENT	Excel Document

AutoCAD	TDM_ACAD_DOCUMENT	ACAD Document
Inventor	TDM_INV_ASSEMBLY	Inventor Assembly
Inventor	TDM_INV_PART	Inventor Part
Inventor	TDM_INV_PRESENTATION	Inventor Presentation
Inventor	TDM_INV_DRAWING	Inventor Drawing
Autodesk Mechanical Desktop	TDM_MDT_ASSEMBLY	MDT Assembly
Autodesk Mechanical Desktop	TDM_MDT_PART	MDT Part
Microsoft Word	TDM_WORD_DOCUMENT	Word Document
SolidEdge	TDM_SE_ASSEMBLY	Solid Edge Assembly
SolidEdge	TDM_SE_PART	Solid Edge Part
SolidEdge	TDM_SE_SHEETMETAL	Solid Sheet Metal
SolidEdge	TDM_SE_WELDMENT	Solid Edge Weldment
SolidEdge	TDM_SE_DRAFT	Solid Edge Draft
SolidWorks	TDM_SW_ASSEMBLY	SolidWorks Assembly
SolidWorks	TDM_SW_PART	SolidWorks Part
SolidWorks	TDM_SW_DRAWING	SolidWorks Drawing
SolidWorks	TDM_SW_PRESENTATION	eDrawing
Pro/ENGINEER	TDM_PROE_REPORT	ProE Report
Pro/ENGINEER	TDM_PROE_MARKUP	ProE Markup
Pro/ENGINEER	TDM_PROE_DIAGRAM	ProE Diagram
Pro/ENGINEER	TDM_PROE_FORMAT	ProE Format
Pro/ENGINEER	TDM_PROE_GROUP	ProE Group
Pro/ENGINEER	TDM_PROE_PART_IAC	ProE Part iAccelerator
Pro/ENGINEER	TDM_PROE_ASSEMBLY_IAC	ProE Assembly iAccelerator
Pro/ENGINEER	TDM_PROE_ASSEMBLY	ProE Assembly
Pro/ENGINEER	TDM_PROE_PART	ProE Part
Pro/ENGINEER	TDM_PROE_DRAWING	ProE Drawing
Pro/ENGINEER	TDM_PROE_MANUFACTURING	ProE Manufacturing
Pro/ENGINEER	TDM_PROE_LAYOUT	ProE Layout
MicroStation	TDM_MI_DOCUMENT	Microstation Document

SmarTeam Integration_Link_Behaviors

The following is a list of **SmarTeam** Integration_Link_Behaviors:

Integration	Integration_Link Behavior	Description
CATIA	TDM_CATIA_COMPOSEDOF	CATIA Composed of
CATIA	TDM_CAT_PRODUCT_LNK	CATIA Product
CATIA	TDM_CAT_DESIGN_LNK	CATIA Design
CATIA	TDM_CAT_RULEBASE_LNK	CATIA Rule Base
CATIA	TDM_CAT_DESIGNTABLE_LNK	CATIA Design Table
CATIA	TDM_CAT_DNSTR_LNK	CATIA Downstream Application
CATIA	TDM_CAT_REF_LNK	CATIA Reference

CATIA	TDM_CAT_CONTEXT_LNK	CATIA Contextual
CATIA	TDM_CAT_RESULT_LNK	CATIA Result
AutoCAD	TDM_ACAD_COMPOSEDOF	AutoCAD Composed of
AutoCAD	TDM_ACAD_IMAGE_LNK	Image
AutoCAD	TDM_ACAD_OVERLAY_LNK	AutoCAD Overlay
Inventor	TDM_INV_COMPOSEDOF	Inventor Composed of
Inventor	TDM_INV_DRAWINGOF	Inventor Drawing of
Autodesk Mechanical Desktop	TDM_MDT_COMPOSEDOF	MDT Composed of
Autodesk Mechanical Desktop	TDM_MDT_TABLE_LNK	MDT Driven table
SolidEdge	TDM_SE_COMPOSEDOF	Solid Edge Composed of
SolidEdge	TDM_SE_DRAFTOF	Solid Edge Draft of
SolidWorks	TDM_SW_COMPOSEDOF	SolidWorks Composed of
SolidWorks	TDM_SW_DRAWINGOF	SolidWorks Drawing of
SolidWorks	TDM_SW_INCONTEXT	SolidWorks In Context
SolidWorks	TDM_SW_DERIVEDPART	SolidWorks Derived Part
SolidWorks	TDM_SW_RAPIDDRAFT	SolidWorks RapidDraft
SolidWorks	TDM_SW_EDRAWING	eDrawing Of
Pro/ENGINEER	TDM_PROEHL_ATTRIBUTES	ProE Attributes
Pro/ENGINEER	TDM_PROE_COMPOSEDOF	ProE Composed of
Pro/ENGINEER	TDM_PROE_DRAWINGOF	ProE Drawing of
Pro/ENGINEER	TDM_PROE_DEPENDENTOF	ProE Dependents
Pro/ENGINEER	TDM_PROE_MANUFACTURINGOF	ProE Manufacturing of
Pro/ENGINEER	TDM_PROELL_ATTRIBUTES	ProE Attributes
Pro/ENGINEER	TDM_PROE_LAYOUTOF	ProE Layouts
Pro/ENGINEER	TDM_PROEL_INSTANCE	ProE Instances
Pro/ENGINEER	TDM_PROEL_SKELETON	ProE Skeleton of
Pro/ENGINEER	TDM_PROEL_EXTREFERENCE	ProE External References
Pro/ENGINEER	TDM_PROEL_MEMBER	ProE Member of
Pro/ENGINEER	TDM_PROEL_SHAREDDRAWING	ProE Shared Drawings
Pro/ENGINEER	TDM_PROEL_DIAGRAM	ProE Diagrams
Pro/ENGINEER	TDM_PROEL_REPORT	ProE Report of
Pro/ENGINEER	TDM_PROEL_MARKUP	ProE Markup of
Pro/ENGINEER	TDM_PROEL_FORMAT	ProE Formats
Pro/ENGINEER	TDM_PROEL_SIMPLIFIEDREP	ProE Simplified Reps
Pro/ENGINEER	TDM_PROEL_GROUP	ProE Groups
Pro/ENGINEER	TDM_PROEL_GROUPMODEL	ProE Group Model
Pro/ENGINEER	TDM_PROEL_IACCELERATOR	ProE iAccelerator
MicroStation	TDM_MI_REFDOC	Microstation Reference Document

=]