Oracle® CADView-3D Synergy API

User's Guide

Release 12

Part No. B40077-01

December 2006

ORACLE®

Oracle CADView-3D Synergy API User's Guide, Release 12

Part No. B40077-01

Primary Author: Reena Faith Titus, Debani Banerjee

Contributing Author: Thomas Godsoe, April Nelson

Contents

Send Us Your Comments

Oracle CADView-3D Synergy API User's Guide, Release 12

Part No. B40077-01

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document. Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

Note: Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the new Applications Release Online Documentation CD available on Oracle MetaLink and www.oracle.com. It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: appsdoc_us@oracle.com

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at www.oracle.com.

Preface

Intended Audience

Welcome to Release 12 of the *Oracle CADView-3D Synergy API User's Guide.*

The introductory section of this manual will be useful to technical managers who are managing projects that will be using Synergy API. This balance of this manual is written for software engineers who will be implementing applications using Synergy API. Before reading this manual, you must have a thorough knowledge of your coding environment:

- Visual Basic
- Visual C++
- VB SCript

or some equivalent system that supports ActiveX components. You must also have experience with programming concepts and practices.

If you are a new user, you should review the introductory material that describes how to use the API. Following that, the API calls are grouped functionally with an overview of the group followed by a detailed description of each element of the group. For experienced users, Appendix C provides a quick reference to the calls, listed alphabetically.

For better clarity, all code samples will be displayed in the following manner: code samples in Courier typeface, slightly indented Names of events, functions and properties will be displayed in boldface. For the purposes of this manual, all references to "application" refer to the program invoking the Synergy API.

See Related Information Sources on page viii for more Oracle Applications product information.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/ .

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Structure

Related Information Sources

- Oracle CADView-3D User's Guide

- Oracle CADView-3D Translator's User's Guide
- Oracle CADView-3D Desktop User's Guide

Integration Repository

The Oracle Integration Repository is a compilation of information about the service endpoints exposed by the Oracle E-Business Suite of applications. It provides a complete catalog of Oracle E-Business Suite's business service interfaces. The tool lets users easily discover and deploy the appropriate business service interface for integration with any system, application, or business partner.

The Oracle Integration Repository is shipped as part of the E-Business Suite. As your instance is patched, the repository is automatically updated with content appropriate for the precise revisions of interfaces in your environment.

Do Not Use Database Tools to Modify Oracle Applications Data

Oracle STRONGLY RECOMMENDS that you never use SQL*Plus, Oracle Data Browser, database triggers, or any other tool to modify Oracle Applications data unless otherwise instructed.

Oracle provides powerful tools you can use to create, store, change, retrieve, and maintain information in an Oracle database. But if you use Oracle tools such as SQL*Plus to modify Oracle Applications data, you risk destroying the integrity of your data and you lose the ability to audit changes to your data.

Because Oracle Applications tables are interrelated, any change you make using an Oracle Applications form can update many tables at once. But when you modify Oracle Applications data using anything other than Oracle Applications, you may change a row in one table without making corresponding changes in related tables. If your tables get out of synchronization with each other, you risk retrieving erroneous information and you risk unpredictable results throughout Oracle Applications.

When you use Oracle Applications to modify your data, Oracle Applications automatically checks that your changes are valid. Oracle Applications also keeps track of who changes information. If you enter information into database tables using database tools, you may store invalid information. You also lose the ability to track who has changed your information because SQL*Plus and other database tools do not keep a record of changes.

1

Introduction of Synergy API

This chapter covers the following topics:

- Purpose of this Manual
- How to Use this Manual
- Conventions
- What is the Synergy API?
- System Requirements
- Installing the Synergy API

Purpose of this Manual

The purpose of this manual is to describe how the Synergy API can be used to integrate the CADView-3D DeskTop with another application.

How to Use this Manual

If you are a new user, you should review the introductory material that describes how to use the API. Following that, the API calls are grouped functionally with an overview of the group followed by a detailed description of each element of the group.

For experienced users, Appendix C provides a quick reference to the calls, listed alphabetically.

Conventions

For better clarity, all code samples will be displayed in the following manner:

```
code samples in Courier typeface, slightly indented
```

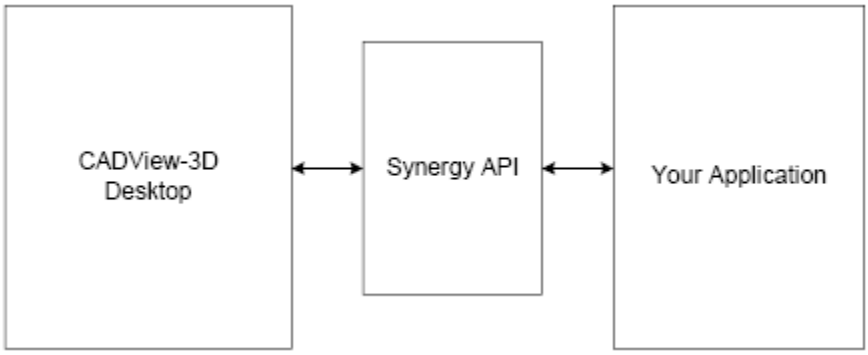
Names of events, functions and properties will be displayed in **boldface**.

For the purposes of this manual, all references to "application" refer to the program

invoking the Synergy API.

What is the Synergy API?

The Synergy API allows your application to communicate with and control theCADView-3D DeskTop.



The Synergy API is an ActiveX control that can be inserted into any application that supports ActiveX controls.

Types of Applications you can Create

The API allows you to develop customized applications for working with your model data. Some examples of the type of applications are:

- **Integrating other database information that is not linked to your model.** Your application can access other databases providing information such as costing, sources, scheduling or materials. Users can then access this additional data as needed. If auxiliary drawings are available, these can be displayed.

- **Customizing the model display.** As requested by the user, you can hide or display selected parts to provide a customized display. For example, users can see objects with or without optional accessories. Or a different color can be used to display parts that are going to be delivered late.

- **Real-time display of processes.** By using color highlights, you can demonstrate how processes will flow through your 3D model.

- **Extended collaboration features**. You can tie your application in to a driver/passenger session. For example, you might display data in a spreadsheet in Microsoft Excel for all collaborators.

Specific Features of the API

The Synergy API is extremely easy to use. Once installed, an application can be talking

to CADView-3D with as few as two lines of source code.

Synergy API provides the following real-time information to your application:

- notification of when parts are picked in the 3D scene
- notification of changes in position and orientation in the 3D scene

Through the API, your application has:

- control over view position and contents of the 3D scene
- limited control over CADView-3D user interface

Based on your application requirements, you can:

- change the color of parts dynamically
- fit on selected parts
- add or retrieve annotations from CADView-3D annotation database
- select parts by name
- select parts by a SQL query
- select parts by spatial proximity
- select parts by many other characteristics
- combine or remove parts from previous selections
- extend the collaboration features of CADView-3D
- retrieve information from the attribute database
- control viewpoints and paths

System Requirements

Use the following information for system requirements.

Application Development System

To develop applications that communicate with the CADView-3D DeskTop, you must have a development system running

- Microsoft Windows 95, 98, 2000, or NT 4 (service pack 3)

- Internet Explorer 4 or higher
- Development PC with sufficient RAM, hard disk space and CPU speed to support the chosen development environment (typically a high-speed Pentium II or Pentium III with 64 MB of RAM and a high-resolution screen)

The Synergy API is an ActiveX control that is language independent. At publication time, CADView-3D supports the following development environments:

- Visual Basic 6.0
- Visual C++ 6.0 with service pack 2
- VBScript

You may also use the Synergy API with any application that supports ActiveX. However, CADView-3D supports only the environments listed.

Client System

In order to use applications that work with Synergy API, the minimum system configuration should be:

- Pentium 133 MHz
- 32 MB RAM
- Windows 95, 98, 2000 or NT 4 (service pack 3)
- Monitor with 16-bit color
- Netscape or Internet Explorer browser

Installing the Synergy API

Synergy API is installed automatically when you install CADView-3D.

2

Using the API

General Concepts and Procedures

Overview of Incorporating the API into your Application

To use Synergy API, simply insert the ActiveX control into your application. You can then issue commands and set up event handlers for the information that you are interested in receiving from CADView-3D. As soon as the CADView-3D DeskTop starts up, you will be communicating with it.

The ActiveX component which forms the API is called **Synergy** and it resides within the **Synergy API** type library. In most cases the name is enough to install the component. Both Visual Basic and Visual C++ provide the user with a list of ActiveX components by name. You simply select that component and use it. Details of how to attach the API to your application in the supported environments are provided in the section about Invoking the API.

Events, Functions and Properties

The Synergy API provides:

- **Events** – generated by CADView-3D and sent to the application. If the application is interested in those events it must define an *event handler* for that particular event. If no handler is defined then the event will never be seen by the application. All events start with:

  On

  For example, **OnPick** and **OnPositionChange** are two events that CADView-3D will generate.

- **Functions –** commands sent from the Synergy API to the DeskTop instructing it to do something. Functions are named according to their task (but will never start with **On**).

- **Properties –** API variables. Properties contain things that are stored within the Synergy API itself, usually configuration information.

Everything in the API falls into one of these categories. In addition any function that needs a return value will cause an appropriate event to be generated.

> **Note:** Function calls will be *queued* inside the Synergy API if the CADView-3D DeskTop is not running, then executed immediately after the DeskTop starts. This allows for an application to pre-configure the DeskTop to its needs, then start the DeskTop directly. A typical use of this would be to switch off toolbars, select some objects and fit them to the screen before starting the DeskTop.

Error Handling

Any function call to CADView-3D may generate an error message. All error messages are returned by an **OnErrorOccurred** event being triggered.

```
OnErrorOccurred(long ErrorCode, BSTR ErrorString,int cookie);
```

The error message has two components: a code and a string. The string is a description of the error, the code is a numeric value that indicates the error. Use **ErrorCode** for fast checking. Use **ErrorString** to present the error to the user.

Errors returned by **OnErrorOccurred** may not be problematic for your application. For instance, it might be a serious error to your application if you attempt to hide items when the DeskTop is not running. Alternatively, you may want to pre-configure the DeskTop by issuing a number of commands before the DeskTop is started. In this case the DeskTop not running is not an error.

Only your application can evaluate the severity of an error condition.

Notation Indicating Parameters with Default Values

A number of the API calls take default parameters. When a value is not supplied for that parameter then the API will use the default one. This is shown in the documentation with a

[default (*VALUE*)]

notation before the parameter name in that function's syntax

description.

For example, the **streamID** value in the API calls usually has a default value of 0. This means that if your model has a single stream then you can avoid defining a stream ID for the API calls.

Notation Indicating Optional Parameters

Some of the function calls have parameters that are optional. For instance, for the **SelectAction** function, the optional parameter **data** needs to be specified for **color** but is not needed for **hide**. If the parameter is marked as optional then you can ignore it if it is not relevant to the action you want to perform. This is shown in the documentation by

[optional]

in front of the optional parameters in the function's syntax description.

Enumerated Types

A number of the API calls take or return enumerated types. These are integers that have a textual representation understood by some ActiveX environments. For instance the **SelectAction** call takes an enumerated type called **ViSelectOperation** which can be one of the following:

```
ViHighlight (=0),
ViFit (=1),
etc.
```

Enumerated types are only valid in some ActiveX environments. Visual Basic and Visual C++ recognize enumerated types, but VBScript does not. For your convenience, this document lists enumerated types both by their names and by their values.

If you are using VBScript, for any function that expects an enumerated type you must substitute the correct integer value. For example, in place of **ViFit** you would use 1, or **ViExamine** would become 6. Failure to do this can lead to elusive errors. For example, the following will work in Visual Basic

```
Synergy.SelectAction ViFit
```

Inside VBScript, **ViFit** would be interpreted as a variable declaration and it would be initialized to 0. So the line would become

```
Synergy.SelectAction 0
```

which would result in the selected objects being highlighted not fitted.

For your convenience, this document lists enumerated types both by their names and by their values.

Streams

CADView-3D allows multiple models to be combined into a single 3D scene. Each model is called a stream. Each model is called a *stream*.

Each stream has a unique attribute and geometry database associated with it. To access geometry (shapes or parts) or attributes you will usually be asked to specify which stream you want to operate on.

Each stream is assigned a unique number when the .AM3 file is loaded by the DeskTop. Stream ID numbers start at 0 and increase by one for each new stream, so if a model has two streams they will be called stream 0 and stream 1. As stream IDs are assigned dynamically at load time, you need to ask the DeskTop how many streams exist and then for a description of each stream after it starts.

Most function calls have the stream ID as a default parameter set to 0, so you only need to specify it if you want to operate on any stream other than the first one.

Shape IDs

Every shape (or part) within CADView-3D has a unique identifier called its shape ID or logical ID (LID). This ID is created by the Optimizer during the optimization phase and is used extensively by the DeskTop and the attribute database. Anytime an attribute is requested, a shape picked, or an object hidden it is the shape ID that is used to describe the object.

If you wish to operate on shapes (or parts) within CADView-3D you will likely need to send LIDs to the DeskTop. You may avoid using LIDs, by using the objects' names or some other attribute instead. However LIDs are the fastest and recommended mechanism. When you refer to an object by its name or attribute, the DeskTop needs to look that information up in the attribute database. This may significantly slow down the processing speed.

There are a number of ways to find out the ID for any shape.

1. The ID is returned when a shape is picked or selected and can be stored and resued later.

2. Any **SelectXXX** command will pick objects, then a**SendSelectedIDs** will return the ids for all objects that the selection matched. **SelectByQuery** and **SelectByName** are good mechanisms for getting shape IDs also.

3. If the attribute database is accessible to the application, then open the database and look up the shape directly. This will be quicker than going through CADView-3D.

Returned Data

Communication between CADView-3D and the Synergy API is done asynchronously. This allows for:

- Limited dependency placed on either application or on the DeskTop

- Improved performance

- Increased flexibility in start up and shut down order.

It does however require a slightly different mechanism than is usual for an API.

Within the Synergy API all function calls return immediately. Any data that is to be returned will come via an event. In the case of an error, the **OnErrorEvent** event will occur.

Not all functions return data. In fact it is a design goal to minimize the amount of data transferred. Wherever possible functions will only return a value if there is an error. If a function triggers an event it is mentioned in the documentation for that function.

For example, if you want to get the name of an object, you must issue the **SendName** function call. Some time later, an **OnName** event (or an **OnError**) event will be sent from the DeskTop.

When an application calls a lot of functions that return data, the application needs to be able to match its requests with the resulting events. This is done by using a *cookie.* A cookie is a unique identifier that the application passes to the DeskTop. When the generates the return event it passes the cookie back along with it. So if the application calls a function with a cookie of 123, any event that returns with a cookie of 123 is the returned data for the request.

Properties

The Synergy API provides a number of properties that can be accessed at any time. The following list describes the properties available.

Property Name	Type	Usage	Description
ErrorCode	Short	Read only	Contains the last error code.
ErrorString	BSTR	Read only	Contains text of the last error.
NumberOfConnections	Long	Read only	Number of connections to the Synergy API.
NumberOfObjects	Long	Read only	Total number of objects in the database.
NumberOfStreams	Short	Read only	The number of streams that the DeskTop is using.

Property Name	Type	Usage	Description
PollRate	Long	Read/Write	Time in milliseconds between Synergy API checks for events.
SourceAm3File	BSTR	Read only	The .AM3 file currently in use.

Response Time

CADView-3D will process API commands and generate events once per update. In the worst case this means you may have to wait up to the time for two updates to complete. If CADView-3D is running fast then this might not be noticeable, but if it is running slowly (e.g., one update every 2 seconds) then it could take several seconds for each API commands to be processed. In addition, if you do anything that requires referring to the collaboration or attribute database then you will also have to wait for the command to be sent to the database, processed and returned to CADView-3D before it can be operated on inside the DeskTop.

You can use the **PollRate** property to specify how frequently Synergy API checks for requests.

Initializing CADView-3D and the Synergy API

This section describes functions and events that are used to determine

- if CADView-3D is running
- how to get information about the currently loaded model such as: how many streams are active, how many objects per stream, and anything else that is model-specific
- how to terminate the application's connection to Synergy API

Functions and Events

Function Name	Description
StartEnvisioni	Start the DeskTop.

Function Name	Description
Flush	Empty queued events from the pending message queue.
Disconnect	End connection with Synergy API.

Event Name	Triggered By
OnClientAlive	The CADView-3D DeskTop starts or another application starts while CADView-3D is already running.
OnClientDeath	The CADView-3D DeskTop terminates.

HRESULT StartEnvisioni(BSTR url)

Parameters

url = the full URL to the .AM3 file to be loaded

Description

This will execute the URL within the default browser. If CADView-3D is installed and the URL points to a .AM3 file then CADView-3D will be started. If the URL does not point to a .AM3 file, or CADView-3D is not installed then the URL will be launched according to the policies of the browser.

HRESULT Flush()

Parameters

None

Description

The Synergy API allows commands to be queued in advance of the DeskTop starting. Thus an application can set up the initial conditions that a user will see when the DeskTop starts up.

By default, if the CADView-3D DeskTop is not running then any function call issued will be queued until the DeskTop starts. Such function calls are called pending messages. When the DeskTop starts, all pending messages are processed in the order that they were received.

Flush will empty queued events from the pending message queue. If your application

wants to queue messages for processing by CADView-3D on startup then this should be the first command issued. Calling this function guarantee that no leftover messages from previous CADView-3D DeskTop sessions remain.

HRESULT Disconnect()

Parameters

None

Description

Close all connections to Synergy API. This call should be the last operation performed.

HRESULT OnClientAlive(BSTR name)

Parameters

name = the .AM3 file that the DeskTop has loaded

Description

This event occurs once whenever the CADView-3D DeskTop starts up or when an application starts using the Synergy API *and* CADView-3D is already running. It passes the name of the .AM3 file loaded in the **name** parameter. The name will contain the full URL.

It is called after the DeskTop has received its initial burst of data. From this point on any API function call can be issued and immediately processed.

> **Note:** If you need to know when to start expecting data from the DeskTop, or just the existence of the DeskTop then install a handler for this event.

HRESULT OnClientDeath(BSTR name);

Parameters

name = the URL that CADView-3D was viewing

Description

This event occurs once as CADView-3D shuts down.

> **Note:** Should be used as a signal to stop sending API calls to CADView-3D. When there are more than one instance of the DeskTop running, the file name can be used to determine which DeskTop terminated.

Controlling the CADView-3D Interface

Synergy API lets you modify how the communication between the API and DeskTop is handled and also how CADView-3D is being used.

Functions and Properties

Function Name	Description
ControlEnvision	Provides limited control over CADView-3D.
DisableUserModification	Disallow users from adding new toolbars.
Toolbar Visibility	Switches the specified toolbar on or off.

Event Name	Contains
PollRate	How frequently the Synergy API and DeskTop talk to each other in milliseconds.

Property PollRate;

Description

Pollrate determines how frequently the Synergy API and the DeskTop communicate with each other (specified in milliseconds). By default this rate is set to 100 milliseconds. If you set it to a lower number then messages will be passed to and from the DeskTop in a shorter time span but more of the CPU will be used by the Synergy API. This may slow down the DeskTop or the user.

Ideally you want this value to be as large as possible before delays between action and response become too large. You are also limited by the fact that CADView-3D will only process commands once per update. No matter how low you set this value you will never get a response quicker than the update rate of the DeskTop.

This can be set at any time.

Examples

PollRate = 500 'update once every ½ second

PollRate = 10 'update 100 times a second

HRESULT ControlEnvisioni(ViControlEnvisioni message, [optional, default(0)]int data);

Parameters

message = what type of control to perform

Description

Provides limited control over CADView-3D. Currently **message** can be set to one of either

- **Pause** (=0)
- **Restart** (=1)

When **Pause** is executed CADView-3D will entirely stop updating. It will not redraw, move the viewpoint, select or show any other action until **Restart** is sent. As this command completely stops the DeskTop it should be used with care, and anytime it is paused, the API writer should remember to **Restart** it at some later time.

Pausing can be useful if you want to free up the CPU to perform some action on your application. CADView-3D consumes a lot of the CPU while it is running so pausing can result in a large increase in responsiveness of other applications.

> **Note:** If you only periodically wanted to move within CADView-3D, then pausing it between movements can be very beneficial.
>
> Also if you want to perform a number of actions and see the visual result only after all the actions have been performed, pausing then restarting will allow that.

Examples

If you wanted to select some objects, color them red, fit them, highlight them you could do the following:

```
ControlEnvisioni Pause
SelectAction Color 255
SelectAction Fit
SelectAction Highlight
ControlEnvisioni Restart
```

HRESULT DisableUserModification(BOOL disable);

Parameters

disable = true or false

Description

If **disable** is set to **true** then the user will not be able to bring up the toolbars menu

when they click the right mouse button. This means that users of the DeskTop can not add new toolbars other than those visible when the **DisableUserModification** command is issued. Setting **disable** to **false** reactivates the menu entry.

> **Note:** Make sure that you have specified the visibility of all toolbars that the user may need before executing this command.
>
> If you use disableUserNotifcaton then it will be possible for the user to switch off toolbars with no way to switch them back on. If this will be a problem for you, you must provide a mechanism, like a button, on the API application that allows the user to switch back on any toolbars that they switched off.

Examples

```
ToolbarVisibility Allbars false
' switch off all toolbars
ToolbarVisibility CollaborationBar true
' switch on the collaboration toolbar
DisableUserModification( true)
' stop them from changing to anything else
```

HRESULT ToolbarVisibility(UserInterfaceToolbars which, BOOL visible);

Parameters

which = toolbar to be set

visible = **true** by default; **false** means switch the specified toolbar off

Description

Switches the specified toolbar on or off according to the **visible** parameter.

Which can be one of

- **AllBars** (=0) : all toolbars
- **StatusBar** (=1)
- **AppearanceBar** (=2)
- **StandardBar** (=3)
- **ObjectBar** (=4)
- **ManagersBar** (=5)
- **ViewpointsBar** (=6)

- **PathsBar** (=7)
- **CollaborationBar** (=8)
- **MovementBar** (=9)
- **RotationBar** (=10)
- **UserBars** (=11) : all bars created by the user

Examples

```
ToolbarVisibility AllBars true
' switch on all toolbars
ToolbarVisibility Allbars false
' switch off all toolbars
ToolbarVisibility CollaborationBar true
' switch on the collaboration toolbar
```

Getting Stream and Model Information

The next group of functions and events allow your application to get information about models (streams).

Functions and Events

Function Name	Description
GetStreamInfo	Returns information for a particular stream: attribute database name, host name, and number of shapes
NumberOfStreams	Returns number of streams that are currently being viewed
SendAttributeTypes	Instruct the DeskTop to send out the attribute types for the currently loaded model.
SendNames	Instruct the DeskTop to send the names of the shapes in the range between and including startID to endID.

Function Name	Description
SendAttributes	Asks the DeskTop to send the attributes for the shape with ID - id on stream = streamID.

Event Name	Triggered By
OnAttributeTypes	SendAttributeType call.
OnNames	SendName received by DeskTop.
OnAttributes	SendAttributes call to DeskTop.

HRESULT GetStreamInfo ([in]int ID, BSTR * dbName, BSTR * dbHost, int * NumberOfShapes);

Parameters

ID = the stream that you want information about

dbName = this will be set to contain the attribute database name of stream [ID]

dBHost = this will be set to contain the name of the host of the database for stream [ID]

NumberOfShapes = set to contain the number of shapes in the stream [ID]

Description

Returns information for a particular stream. Use **NumberOfStreams** to find out how many streams exist.

> **Note:** Stream IDs start at 0, so if there are 4 streams these are identified as 0,1,2, and 3.

HRESULT NumberOfStreams([out] short *count);

Parameters

count = the number of streams in the .AM3 file that CADView-3D is currently viewing.

Description

Returns the number of streams.

Note: Stream IDs start at 0, so if there are 4 streams these are identified as 0, 1, 2, and 3.

HRESULT SendAttributeTypes([default = 0] int which);

Parameters

which = which stream that you want information for. If you do not supply a value then the attributes for stream 0 will be sent.

Description

Instruct the DeskTop to send out the attribute types for the currently loaded model.

Note: The DeskTop will send out all the attribute types for the requested stream for the currently loaded model. The information will come back in the form of an **OnAttributeTypes** event. You must define a handler for this event to extract the data.

HRESULT SendNames([optional, default(0)]int streamID,int startId, int endId,[optional, default(0)]int cookie);

Parameters

streamID = the stream of interest (0 = default value)

startID = the shape ID that the listing is to start

endID = the shape ID that the listing is to end at

cookie = optional identifier can be used to tally with returned events

Description

This function instructs the DeskTop to send the names of the shapes in the range between and including **startID** to **endID**. Any IDs that do not exist in the specified stream are ignored. This command will cause an **OnNames** event to be generated when the result is returned. You must have an event handler for **OnNames** defined to see any results from this call.

If you make multiple send requests then use the **cookie** parameter to specify a unique ID for each request. The **cookie** value is passed to the DeskTop who will then return it in the **OnNames** event unchanged. This allows the caller to identify requests with responses. It is the applications responsibility to provide unique values.

Note: Make sure that the IDs exist before sending or matching the returned strings will be difficult.

If you wanted to get the names for all objects in a stream then use a

startID of 1 and an **endID** of the number of objects returned in a **GetStreamInfo** call. If you do ask for all names in a stream be very conscious of the number of elements in the stream. If there are a million objects in the stream then this may take a very long time to complete, and may seriously affect the free memory on the DeskTop machine. Only ask for everything if you are sure that the data to be returned will be small. This is true of any function which returns data from the CADView-3D DeskTop.

HRESULT SendAttributes(int streamID, int id,[optional, default(0)] int cookie);

Parameters

streamID = stream ID that the query is being made on (default value is 0)

id = the shape ID that you want attributes for

cookie = a user supplied value that can be used to tally commands with returned events.

Description

Asks the DeskTop to send the attributes for the shape with ID = **id** on stream = **streamID**. The DeskTop will gather the information from the attribute database and then trigger an **OnAttribute** event. If you are asking for the attributes for more than 1 shape then the cookie value can be used to tag a send with the corresponding event it generates.

Note: See **OnAttributes** for the format of the returned data array.

HRESULT OnAttributeTypes(VARIANT names);

Parameters

names = array of strings containing the names of all attributes

Description

The DeskTop triggers this event in response to a **SendAttributeTypes** call. The array contains the names of all attribute types for the stream requested. The index into the array corresponds to the Attribute type ID (starting from 1) for that string. For example, if the array contained

"layer","weight","height"

Then there are three attribute types in the attributes database with IDs 1 = "layer", 2 = " weight", 3 = "height".

If you want to request an attribute of a particular type then you must use the index into this array.

HRESULT OnNames(int streamId, int startID, int endID,VARIANT names, int cookie);

Parameters

streamID = which stream this information relates to

startID = the starting ID of the shape names being returned

endID = the end ID of the shape names being returned

names = array of strings

cookie = a user supplied value used to tally commands with returned events

Description

This event is triggered whenever the DeskTop receives a **SendNames** event. The array returned contains the names for each of shapes contained within the start and end IDs inclusive. If **startID** = 2 and **endID** = 4 then the array would contain 3 entries for the shapes 2, 3, and 4.

The **cookie** value can be used to tie this event to the **SendNames** event that generated it, however as the IDs for a stream are unique you should never need to use a cookie for this command.

HRESULT OnAttributes(int streamID, int lid, VARIANT names, VARIANT values)

Parameters

streamID = the stream that this data is from

lid = the shape ID that this data is for

names = array of strings, the attribute types that have been returned

values = array of strings, the actual attribute data

Description

The DeskTop sends this event in response to a **SendAttributes** call. For each entry in the **names** array there is a corresponding entry in the **values** array. The **names** entry specifies what the attributes type is, and the **values** entry specifies what the actual value of the attribute is.

For instance, if the arrays contained….

names = {"layer","color","name" }

values = {"1","red","dougal"}

Then these would be reconstructed as Layer = 1 ; color = red; name = dougal

Selecting Parts and Operating on that Selection

One of the most common operations within the API is the requirement to select parts

within the database and then operate on those parts. The API provides a large number of ways to select shapes, described below.

General Procedure

Within the Synergy API you:

1. Select one or more parts.
2. Execute an action on that selection.

Selection by itself causes no visible effect on the DeskTop. It is the action following the selection that causes something to change. A selection will persist until a **SelectClear** call is issued or the user of the DeskTop manually selects something.

The selection mechanism is designed to work on small, medium, and large data sets, with the minimal transmission of data but with the maximum flexibility.

You can hide, show, color, isolate, examine, fit, and highlight a selection. The selection process itself is very flexible, there are numerous ways to select objects and selections using different criteria can be combined with each other to produce very specific results. For instance, it is possible to, for example, select all objects with an attribute X=Y, that are within 5 meters of object Z, and then remove any which are in Layer=W.

The Selection Modifier

Multiple selection criteria are established by making more than one selection function call. Every call takes a **ViSelectModifier** which can be one of

Clear (=0) – erase any previous selections, and make the results of this operation the current selection. This is the default value.

Add (=1) – add the results of this selection to the current selection

Remove (=2) – remove the results of this selection from the current selection

So by issuing multiple selection calls with the **Add** modifier then all results will be combined.

For example,

```
SelectByName("one",Add)
SelectByName("two",Add)
SelectById(0,lids,Add)
```

would select all shapes called "one", or "two" and all shapes within the LIDs array.

If the following was then issued

```
SelectByName("another shape") or
SelectByName("another shape",Clear)
```

that would erase the previous selections and only select objects called "another shape".

The command

```
SelectByAttribute("layer","0",Remove)
```

would remove all shapes that are in layer 0 from the current selection.

The Current Selection

All **SelectByXXXXX** calls operate the same way: they add their results to what is known as the *current selection*. The current selection is a list of shapes. Once you have performed a selection you can request feedback from the DeskTop about the current selection. With this feedback you can then refine your search or validate that you have in fact picked the correct things.

Specifying the Action to Perform on the Selection

Selecting shapes causes no visual change to the 3D scene. To cause something to happen you must issue a **Specifying the Action to Perform on the Selection** call with the appropriate action that you want performed. You can issue as many **SelectAction** calls that you wish and they will be executed in the order specified. For instance, if you wanted to highlight and then fit on the current selection you would do the following

```
SelectAction ViHighlight
SelectAction ViFit
```

Changing the Selection

After you make a selection it will remain unchanged until either

- you make another selection (modification based on the value of **ViSelectionModifer**)
- you issue a **SelectClear** call
- the user picks something in the 3D scene

Caveats

Some points to be aware of when making a selection:

1. Be careful when making general queries on large data sets. If you do a query which matches all objects in a million shape model then any action could be very slow. Try to limit the number of shapes that will match a selection if possible, or do a reverse selection where you select everything then remove things you do not want.
2. Selecting by shape ID is much faster than any other kind of selection. Do this whenever possible.
3. **SelectByName**,**SelectBySQL**,**SelectByAttribute** all require a query to the attribute

database to get the results. This can be slow if the data set is large and the queries match a lot of objects.

Functions and Events

Function Name	Description
SelectByName	Will select any parts on the DeskTop that have a name that incluses the string within the "name" parameter.
SelectById	Select all the shapes that have the Ids passed in the LIDs array on the stream specified by StreamID.
SelectAll	Select all shapes that are inside this stream.
SelectByQuery	Searches the attribute database for any attributes with the name specified that match the value.
SelectSpatially	Select all shapes that fall within a distance from a shape, collection of shapes, or a point in space.
SelectByIdRange	Select a range of shapes by using the start and end Ids for the range of objects.
SelectBySQL	This will execute the SQL statement you specify on the attribute database of the specified stream.
SelectClearAll	Clears the current selection on the DeskTop.
SelectAction	This will perform the specified action (highlight, fit, hide, show, etc.) on the current selection.
SetHightlightMode	Specify what the highlight action will do.
SendSelected	Inform the DeskTop that you want to know information (e.g. number of shapes) about the current selection.

Event Name	Triggered By
OnPick	The user picks one or more shapes.
OnSelectedCount	Calling **SendSelection ViReturnCount**
OnSelectedIds	Calling **SendSelection ViReturnIds**
OnSelectedData	Calling **SendSelection ViReturnData**

HRESULT SelectByName(BSTR name, [optional, default(Clear)]ViSelectionModifier mod);events

Parameters

name = the substring that you want to search for

mod = the selection modifier

Description

Will select any parts on the DeskTop that have a name that includes the string within the **name** parameter. Case is not significant with this match.

> **Note:** This requires the DeskTop to do a string comparison on the attribute database and can be slow when executed on very large models.

Examples

```
SelectByName "cover", Clear
SelectByName "over"
```

HRESULT SelectById([optional, default(0)]int StreamId, SAFEARRAY(int) *LIDs, [optional, default(Clear)]ViSelectionModifier mod);

Parameters

StreamID = the stream that this selection is to be applied to

LIDs = integer array of the ID values of the shapes to be selected

mod = the selection modifier

Description

Select all the shapes that have the IDs passed in the **LIDs** array on the stream specified by **StreamID**. If no shape exists for a supplied ID then that value will be ignored

Note: This is one of the quickest ways to select shapes.

HRESULT SelectAll([optional, default(0)]int StreamID,[optional, default(Clear)]ViSelectionModifier mod);

Parameters

StreamID = the stream to operate on

mod = the selection modifier

Description

Select all shapes that are inside this stream.

If you want to select all shapes on all streams, set **StreamID** = -1

> **Note:** If you want to select most shapes in a model it is usually more efficient to do a **SelectAll** and then remove the entries that you do want by doing a **SelectXXX** call with the selection modifier set to **Remove**. For instance, if you wanted all shapes but those on layer 1 do

```
SelectAll –1
SelectByAttribute "layer","1", Remove
```

Examples

```
SelectAll 'select all shapes in stream 0
SelectAll 0  ' select all shapes in stream 0
SelectAll 1 ' select all shapes in stream 1
SelectAll –1 ' select all shapes in all streams
```

HRESULT SelectByQuery(BSTR AttributeName, BSTR AttributeValue, [optional, default(Clear)]ViSelectionModifier mod);

Parameters

AttributeName = the name of the attribute to query against

AttributeValue = the value that you want to match

mod = the selection modifier

Description

Searches the attribute database for any attributes with name specified by **AttributeName** that match the value specified by **AttributeValue**. This is a string comparison and as such the SQL wild cards characters of '_'(match single character) and '%' (match all) can be used in the **AttributeValue** string. For instance

```
SelectByQuery("Layer","one")
```

will select all shapes that have an attribute called Layer that contains "one".

The command

```
SelectByQuery("Layer","o_e")
```

will select any shapes that have an attribute called Layer that has a value with 3 letters starting with 'o' and ending with 'e'.

The command

```
SelectByQuery("Layer","o%e")
```

will select any shapes that has a Layer value that starts with a 'o' and ends with an 'e'. So

- one
- onlyone
- ohhhe

would all match.

> **Note:** The **AttributeName** value must match exactly to the name of the Attribute type specified in the attribute database.
>
> **Case is significant.**
>
> You can get all the names of the attributes by issuing a **SendAttributeTypes** call.

HRESULT SelectSpatially(float x, float y, float z, float radius, [optional, default(0)] int streamID, int lid, SpatialPickType type, [optional, default(Clear)]ViSelectionModifier mod);

Parameters

x = the x coordinate of the center point

y = the y coordinate of the center point

z = the z coordinate of the center point

radius = the size in database units of the picking sphere

streamID = optional argument that is used only when the lid is

greater than 0

lid = modifier for the type of pick

type = the kind of spatial pick to be performed.

mod = the selection modifier

Description

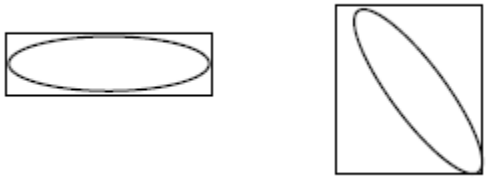
Select all shapes that fall within a distance from a shape, collection of shapes, or a point in space.

The type of pick can be:

- **EntirelyWithin** (=1). Pick only shapes which are entirely within the radius
- **WithinOrTouching** (=0). Pick any shape which is within or touches the radius

The spatial pick is done against the bounding box of a shape unless some geometry has already been downloaded. In that case the spatial pick will pick against the geometry of that shape and be much more accurate. Due to the dynamic nature of the DeskTop, where it loads and deletes shape data as necessary it is not always possible to know if geometry has been downloaded or not. It is safest to assume that picking will be done against bounding box and be correspondingly inaccurate. You may want to make the radius larger than actually wanted to get all required shapes.

Note on Bounding Boxes: A bounding box is the smallest axially aligned box that will completely contain a shape. This box is always bigger than the shape itself, and in some cases where the shape is non symmetrical and not aligned with the x,y,z axis, the error in size can be huge.



In the left image above, the bounding box is fairly close to the original. In the right image, the same shape deviates from the axis and the bounding box is almost 3 times the size of the other.

Controlling the Spatial Pick with the Lid Parameter

Use the following steps to select the lid value of the shape.

To select from a shape

Set the **lid** value to the ID of the shape that you want to do the spatial pick from.

The **streamID** specifies the stream to which the lid belongs.

The **x, y** and **z** values represent an offset from the center of the bounding box of this shape. A value of (0,0,0) would mean use the exact center. A value of (0,0,-10) would mean pick from the center of the bounding box offset by 10 database units in the z coordinate.

For example,

```
SelectSpatially 0,0,0,1.0,0,34,EntirelyWithin,Clear
```

would select all shapes that are within 1 database unit of the shape with an ID = 34 in stream 0.

To use the current selection

Set the **lid** parameter to -1.

This instructs the spatial pick to use whatever the current selection is to determine the center point for the spatial query.

If multiple shapes are in the current selection then the center of their combined bounding box is used. If one shape is selected then its center point is used. The **x, y** and **z** values are used as an offset from this center point.

For example,

```
SelectByName "cone"
SelectSpatially 0,0,10,1.0, -1,EntirelyWithin,Clear
```

would use the results of the "cone" selection, generate the center point of all the current selection, offset it by 10 database units in the z coordinate, then pick all shapes which fall within 1 units of that point.

To use a point in space

If you just want to pick all shapes from a particular point in space, set the **lid** value to 0 and specify the point you want in the **x, y,** and **z** values. For example,

```
SelectSpatially 100,10,25,1.0,,0,EntirelyWithin,Clear
```

To pick objects outside a certain distance

If you want to pick objects that are greater than a particular distance from a shape then use the **Remove** modifier in combination with **SelectAll** in the following way:

```
SelectAll -1 // select all shapes in all streams
SelectSpatially 0,0,0,10.0,2,45,EntirelyWithin,Remove
```

The first line selects all objects, the second selects all objects that are within 10 units of shape 45 in stream 2 and then *removes* them from the selection. Any shapes that are left are those that failed the spatial select and are therefore further away than 10 database units from shape 45 in stream 2.

HRESULT SelectByIdRange([optional, default (0)] int streamID, int startLID, int endLID, [optional, default(Clear)] ViSelectionModifier mod);

Parameters

streamID = the stream that this selection is to operate on

startLID = the ID of the start Shape

endLID = the ID of the last shape

mod = the selection modifier

Description

Select a range of shapes by using the **startID** and **endID** for the range of objects. This does the same as **SelectByID** except in a more efficient manner. By only specifying the start and end IDs, no lengthy array needs to be built and passed to the DeskTop. If you want to select a large number of shapes (e.g., 500,000) this is the way to do it. If no entry exists for an ID then it is ignored.

For instance, if your model has shapes with IDs 30 through 50, and 55 through 60 then the command

```
SelectByIdRange(0,30,60)
```

would pick all objects, ignoring the discontinuity in the ID numbers.

> **Note:** Usually IDs start at 1 and there is 1 ID for each shape in the model. Sometimes however there are gaps in the ID range generated by the optimizer. These can be safely ignored.
>
> This is the quickest and most memory-efficient way to pick shapes.

HRESULT SelectBySQL(int streamID, BSTR SQL,[in] [optional, default(Clear)] ViSelectionModifier mod);

Parameters

streamID = the stream that you want to be queried

SQL = the SQL statement that you want issued

mod = the selection modifier.

Description

This will execute the SQL statement you specify on the attribute database of the specified stream. Any shapes that match the query will be selected. In order to select objects, the SQL query must be of the form

```
Select * from Attribute <query>
```

where <query> is your SQL statement.

> **Note:** You must understand both SQL and the CADView-3D attribute database format to use this command. No error checking is performed on the SQL string.

HRESULT SelectClearAll();

Parameters

None

Description

Clears the current selection on the DeskTop.

HRESULT SelectAction(ViSelectAction action, [optional, default(0)]int data);

Parameters

action = the operation to be performed

data = optional data field that is required by some actions

Description

This will perform the specified action on the current selection. The selection will remain unchanged.

Action can be one of

- **ViHighlight** (=0) – Perform the active highlight mode on the current selection (see SetHighlightMode for details). Default action is to switch the selected objects to wireframe (**viWireframe**)
- **ViFit** (=1) – Move the viewpoint until the current selection fits onto the screen. Once on screen, the position is moved as close as possible to the selection while still keeping it on screen.
- **ViHide** (=2) – Hide the selection from the view.
- **ViShow** (=3) – Draw the selection on the view. This is the opposite of **ViHide.**
- **ViIsolate** (=4) – Hide everything but the selection and fits on the selection.
- **ViUnisolate** (=5) – Undo the last isolate. Viewpoint and visibility are returned to the way they were.
- **ViExamine** (=6) – Isolate the selection and switch on examine mode. This lets you rotate around the selection while keeping it centered on screen.
- **ViColor** (=7) – Change the color of the selection. The color value is specified in the data parameter in the standard packed color form of 0xggbbrr. A value of 0xff(255) = red, 0xff00 (65280) = blue, and 0xff0000(16711680) = green.
- **ViUnhighlight** (=8) – Undo of highlight. Makes sure that the selection does not have the highlight on.
- **ViUnexamine** (=9) – Undo of examine. Restores visibility and view position to the way it was before and reverts to the standard movement mode.
- **ViDefaultColor** (=12) – Restore selection to default color. Any shape that has had its color changed via the **ViColor** action will be restored to its true color.

Note: A very common mistake is to forget that VBScript does not support enumerated types. For example,

```
SelectAction ViFit
```

works fine in Visual Basic but in VBScript it will be interpreted as

```
SelectAction 0
```

If you are using VBScript use the numeric value shown in the list above in place of the enumerated type.

```
SelectAction 1
```

would perform a fit action.

Examples

```
SelectAction Fit
```

HRESULT SetHighlightMode(ViHighlightMode am);

Parameters

am = the highlight mode

Description

Sets what the affect the highlight action will have. This can be one of:

- **ViWireframe** (=0) - Highlighted elements are shown in wireframe.
- **ViWireOverSolid** (=1) - Highlighted elements are shown in wireframe over the solid shape.
- **ViSolid** (=2) - Show selection in solid view everything else in wireframe.
- **ViBoundingBox** (=3) - Show the bounding box of the selection.

HRESULT SendSelected(ViSelectReturn which, [optional, default(0)] int cookie);

Parameters

which = the type of information you want returned

Description

Inform the DeskTop that you want to know information about the current selection.

Which can be one of

- **ViReturnCount** (=1) – Request the number of shapes in the current selection. This will cause an**OnSelectedCount** event to be triggered.
- **ViReturnIds** (=2) – Request the IDs of the current selection. This will trigger an

OnSelectedIds event.

- **ViReturnData** (=4) – Request bounding box information about the selection. This will trigger an **OnSelectedData** event.

In order to process the returned information you must have a handler defined for the event that is returned.

> **Note:** If you are dealing with large databases it is usually advisable to first request the count, and check to see that not too many shapes are present before requesting the ids or data. These arrays can become very large and may adversely affect performance.

Examples

The commands

```
SelectByAttribute("Layer","5")
SendSelected ViReturnCount
```

will return the number of shapes that the layer = 5 query selected.

void OnPick(BSTR name, int streamID, int handle, int count);

Parameters

name = name of the object picked

streamID = the stream that the object belongs to

handle = the ID of the picked object

count = the total number of shapes picked

Description

This is an event generated by the DeskTop when the user picks one or more shapes. You get the name and id information of the shape. In the case where the user has picked more than one shape you get the last shape picked. If the count value is greater than 1 then use the **SendSelected** function calls to get more information on the user's selection.

> **Note:** You must install this as an event handler before it will get triggered.

HRESULT OnSelectedCount(int count);

Parameters

count = the number of shapes in the current selection

Description

This returns the number of shapes in the current selection and is triggered by calling the **SendSelection ViReturnCount** function.

> **Note:** You must install this as an event handler before it will get triggered.

HRESULT OnSelectedIds(VARIANT ids);

Parameters

ids = array of integers representing the ids of the current selection

Description

Sent when the DeskTop receives **SendSelection ViReturnIds**.

The array is formatted to implicitly encode the stream each selection resides in. Starting from stream 0 the selected shapes for that stream are listed, and then terminated by a 0, then stream 1 starts and so on for all streams or until the array is emptied.

For example

1, 3, 6, 0, 2, 3,45, 0, 22, 94

would be

stream 0 = 1,3,6

stream 1 = 2,3,45

stream 2 = 22,94

> **Note:** You must install this as an event handler before it will get triggered.

HRESULT OnSelectedData(VARIANT ids, VARIANT boxMin, VARIANT boxMax);

Parameters

ids = integer array of the ids of the selected shapes

boxMin = float array of the lower left corner of the bounding boxes

boxMax = float array of the upper right corner of the bounding box

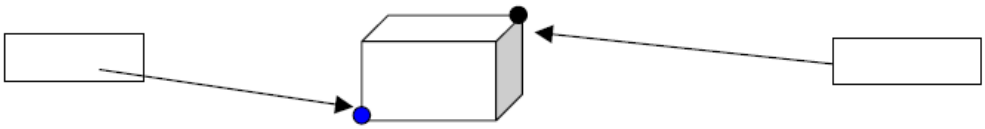
Description

This event is triggered when the DeskTop receives

```
SendSelection ViReturnData
```

This event is provided with the IDs of the current selection and the bounding boxes for those IDs. The bounding box is specified by 6 floating point numbers, as shown in the diagram below:

- **boxMin** specifies the X,Y,Z coordinates of the lower left corner of the bounding box.
- **boxMax** specifies the upper right corner of the bounding box.



The arrays are formatted to encode the stream each selection resides in. Starting from stream 0 the selected shapes for that stream are listed, and then terminated by a 0, then stream 1 starts and so on for all streams or until the array is emptied. For example the IDs array containing

1, 3, 6, 0, 2, 3,45, 0, 22, 94

would be interpreted as

stream 0 = 1,3,6

stream 1 = 2,345

stream 2 = 22,94

The same encoding method is used for the bounding box data except that each entry has three values for (x,y,z). The bounding box information is then combined with the LID information. For example if **boxMin** contained

0, 0, 0, 2,2,2, 1,1,1 ,0 *(end of stream marker)*, 3,3,3, 2,1,2, 5,4,3, 0, 7,3,2, 9,4,0

and **BoxMax** was

1,1, 1, 3,3,3, 4,4,1 ,0 *(end of stream marker)*, 7,0,3, 2,6,2, 2,9,4, 0, 2,4,1, 8,2,1

this would be decoded as the following:

Stream 0

Shape 1 = {0,0,0} , { 1,1,1}

Shape 3 = {2,2,2}, {3,3,3}

Shape 6 = {1,1,1} {4,4,1}

Stream 1

Shape 2 = {3,3,3} { 7,0,3}

Shape 3 = {2,1,1} { 2,6,2 }

Shape 45 = {5,4,3} { 2,9,4}

Stream 2

Shape 22 = {7,3,2} { 2,4,1}

Shape 94 = {9,4,0} {8,2,1}

Setting or Retrieving Position Information

Users of the Synergy API can set the view position and orientation separately, and receive feedback from CADView-3D when the user moves the view around.

The API also lets the application switch between examine and drive movement modes.

Functions and Events

Function Name	Description
SetPosition	Sets the view position to the value specified.
SetOrientation	Specify the orientation in degrees.
SetMovementMode	Switches the DeskTop between two supported movement modes.

Event Name	Triggered By
OnPosition	The user's view position moves.
OnOrientation	The user changes their orientation within the model.

HRESULT SetPosition(float x, float y, float z);

Parameters

x = x coordinate for the view position

y = y coordinate for the view position

z = z coordinate for the view position

Description

Sets the view position to the value specified by **x**, **y**, and **z**, specified in database units.

HRESULT SetOrientation(float h, float p, float r);

Parameters

h = heading

p = pitch

r = roll

Description

Specify the orientation in degrees. The orientation Euler matrix is built by combining heading, pitch, and roll in that order. The angles themselves should be in the range –180 to 180.

When this function is called, the DeskTop is immediately set to the orientation contained within **h, p,** and **r**.

> **Note:** Euler angles do not always combine intuitively. If in doubt then only set the pitch and heading values, leave roll at 0.0. By omitting roll you should always see what you expect.

HRESULT SetMovementMode(int value);

Parameters

value = movement mode

Description

Switches the DeskTop between the two supported movement modes.

Examine (=1) - Rotate around the currently selected shapes.

Drive (=0) - "Drive" the eye point through the database as though you were in a car.

void OnPosition(float x, float y, float z);

Parameters

x, **y** and **z** = the current view position in the DeskTop

Description

This event is triggered by the DeskTop anytime the view position moves.

> **Note:** Changes in orientation will not trigger the event, only position. Set up a handler for **OnOrientation** if you are interested in the current orientation.

void OnOrientation(float h, float p, float r);

Parameters

h = heading angle

p = pitch angle

r = roll angle

Description

Event that is triggered by the DeskTop every time the user changes their orientation within the model. Specify **h**, **p** and **r** in degrees in the range –180 to 180.

Collaboration

The Synergy API allows for extension of the collaboration facilities offered by CADView-3D. There are numerous applications for this. The simplest would be to synchronize user interface actions of the driver to the user interfaces of all the passengers for your application. You could make dialogs pop up, buttons depress, text appear, icons change, all apparently caused by the driver of the session.

Users of the API get notification of the start of any session, whether they are a passenger or driver, and the ability to send arbitrary data message to other DeskTops (running with the same Synergy API application). The DeskTop deals with the routing of the messages to the participants of the session. All the Synergy API application need do is to define what the data to be sent is and then call **SendCollaborationMessage**. If you are a passenger and you call **SendCollaborationMessage** then the message is sent only to the driver. If you are the driver then the message is sent to all passengers. Users of the API are free to send any type of data, but they are responsible for encoding the data, and deciding on what that data means.

There are two events that can be trapped **OnCollaborationMessage,** and **OnCollaborationStateChange**. The first is received as the result of a call to **SendCollaborationMessage** by someone in the collaboration session, the second is triggered whenever the DeskTop becomes a driver or a slave. The combination of these events with the ability to send data over the network to others in a session is a very powerful feature.

Functions and Events

Function Name	Description
SendCollaborationMessage	Passengers: sends the data to the driver. This allows passengers to talk to the master. Drivers: sends the message to all passengers in the session.

Event Name	Triggered By
OnCollaborationMessage	Someone in the collaboration session issuing a **SendCollaborationMessage** call.
OnCollaborationStateChange	Whenever a change in the Collaboration State of the DeskTop occurs.

HRESULT OnCollaborationMessage(int MessageType, VARIANT d);

Parameters

MessageType – application defined

d = array of bytes – application defined

Description

This event is triggered when someone in the collaboration session issues a **SendCollaborationMessage** call. If you are currently the driver then this has come from a passenger. If you are a passenger then you know it has come from the driver of the session. It is up to the application to define what the message type and data represent. In the simplest type of communication, **MessageType** could be a simple instruction. For example, if **MessageType** = 2 might mean depress the user interface button and **MessageType** =3 might mean release the button. Then the driver of a session could very easily cause all passengers to modify the user interface of their application to mimic what was happening on the driver. If it was a Web page, you could click a button on the page of all passengers.

HRESULT SendCollaborationMessage(int messageType, [optional] SAFEARRAY(unsigned char) *data);

Parameters

messageType = application defined value

data = application defined byte array

messageType can be any value, **data** can be any size and contain anything that the application requires.

Description

Calling this function when you are a passenger will send the data to the driver. This allows passengers to talk to the driver. Calling this function when you are a driver will send the message to all passengers in the session.

The messageType and data are passed unchanged into the **OnCollaborationMessage** event.

Note: Calling the function when no collaboration session is in place will result in the message being discarded.

It is possible to send anything between applications. Whole documents could be sent if required. It is up to the API user to resolve any security issues this may cause. By itself this is not a security risk, as CADView-3D will not automatically send any data. The Synergy API user must determine what can be sent and provide the encoding and decoding mechanism. It is not possible for one rogue person to write a Synergy API application to steal data from elsewhere. All connected users must be running the same Synergy API application in order to make sense of the data, and make the communication work.

It is recommended that when a session starts, the driver of the session sends out a protocol value in messageType that all passengers can receive and validate.

For example

```
Synergy_OnCollaborationStateChange(ViCollaborationState which)
Begin sub
dim data(1) as integer
data[1] = 1234 ' 1234 = protocol version number
' send out the protocol message (messageType = 1
if which = ViDriver then
SendCollaborationMessage 1,data ' sends this to all
passengers in the session
ProtocolOk = true ' masters protocol is always ok
endif
end sub
Synergy_OnCollaborationMessage( int MessageType, VARIANT d)
Begin sub
if messageType = 1 then
if d(1) = 1234 then
protocolOK = true
else
protocolOK = false
endif
else
if protocolOK then
'process any other messages
endif
endif
```

```
end sub
```

HRESULT OnCollaborationStateChange(ViCollaborationState which);

Parameters

which = what the change was

Description

This event is triggered whenever a change in the collaboration state of the DeskTop occurs. The state will change when the session starts, if a transfer of control happens, and when you leave the session.

The change can be:

ViDriver (=2) – This DeskTop is the driver.

ViPassenger (=1) – This DeskTop is a passenger.

ViFree (=0) – This DeskTop has left the collaboration.

Annotation

The Synergy API provides limited control over the annotation database used by CADView-3D. Users of the API can

- Store annotations into the database.
- Use any JPEG image as the background for the annotation.
- Associate that image with text and/or a URL.
- Retrieve annotations from the database. When retrieved the user gets the composite image built from the background plus any 2D information that was added to the image.

Using this mechanism allows for API users to store arbitrary image data into the annotation database. Once in the database it can be marked up in the same way as any other annotation.

Functions and Events

Function Name	Description
ExtractAnnotationImage	Retrieves an annotation image from the annotation database.

Function Name	Description
InsertAnnotation	Creates an annotation inside the annotation database.
SendAnnotations	Instructs CADView-3D to send a list of the names of all known annotations for the currently loaded model. This will generate an OnAnnotations event.
ViewAnnotations	Forces CADView-3D to display the annotation of "name". CADView-3D will switch to annotation mode and display the annotation if it exists. If the name is duplicated then the first name found will be used.

Event Name	Triggered By
OnAnnotationReturn	Either an **InsertAnnotation** or an **ExtractAnnotation** call
OnAnnotations	Return from a SendAnnotations request. Contains a list of all names of annotations known to CADView-3D
OnAnnotationChange	Whenever an annotation is added or deleted to the collaboration database. name = the name of the annotation, ViAnnotationChange = ViAnnotationAddition or ViAnnotationDeletion, user = the login name of the person who made the change. Users can then do an ExtractAnnotation if they want more information.

HRESULT ExtractAnnotationImage(BSTR AnnotationName, BSTR filename, [optional, default(0)] int cookie);

Parameters

AnnotationName = the name of the annotation to be retrieved

Filename = full path to where the image is to be stored

cookie = optional value to track for errors

Description

Retrieves an annotation image from the annotation database and stores that as a JPEG file on the local system in the location specified by **filename**. The image will be the composite image built from the background image plus any 2D information that the creator of the annotation added.

AnnotationName must match exactly the name of the annotation required. If no annotation matches this name then the **OnAnnotationReturn** event will be triggered with an error value. Otherwise **OnAnnotationReturn** will get triggered with a status value greater than zero.

> **Note:** The annotation must exist in the model that the DeskTop is currently viewing.

HRESULT InsertAnnotation(BSTR name, BSTR imagelocation,[optional, default(" BSTR url,[optional, default(" BSTR text, [optional, default(0)] int cookie);

Parameters

name = what the annotation is to be called

imagelocation = full path to the jpeg image

url = URL to be associated with the annotation

text = text to be associated with the annotation

cookie = optional tracking parameter

Description

Creates an annotation called **name** inside the annotation database for the model currently being viewed by the DeskTop. You can optionally specify a URL or text values that will also be stored with the annotation. If you omit these values then the annotation will be created with no entries for these things.

If the addition is successful then **OnAnnotationReturn** will be triggered with status >0 and **cookie** equal to the **cookie** parameter passed into **InsertAnnotation**.

HRESULT SendAnnotations();

Parameters

Description

Instructs CADView-3D to send a list of the names of all known annotations for the currently loaded model. This will generate an OnAnnotations event.

HRESULT ViewAnnotation(BSTR name);

Parameters

Description

Forces CADView-3D to display the annotation of "name". CADView-3D will switch to annotation mode and display the annotation if it exists. If the name is duplicated then the first name found will be used.

HRESULT OnAnnotationReturn(int status, int cookie);

Parameters

status = contains information about an annotation addition

cookie = tracking value. This is set to equal the value passed in

Description

Triggered by either an **InsertAnnotation** or an **ExtractAnnotation** call.

Status can be one of the following:

>0 operation was successful

-1 = annotation not found

-2 = annotation name not found

-3 = no collaboration database found

HRESULT OnAnnotations(VARIANT name)

Description

Is triggered by a SendAnnotations request.

Name = variant array of Strings representing all names of annotations known to CADView-3D.

If there are no annotations for the loaded model then the list will be empty.

HRESULT OnAnnotationChange(BSTR name, ViAnnotationChange which, BSTR user);

Parameters

name = the name of the annotation that changed

which = ViAnnotationAddition or ViAnnotationDeletion

user = the login name of the person who made the change

Description

Is triggered whenever an annotation is added to or deleted from the collaboration database. Status can be one of the following:

Users can then do an ExtractAnnotation if they want more information.

Viewpoints and Paths

The Synergy API allows users to retrieve a list of all viewpoints and paths in the currently loaded model inside CADView-3D . Once you have the lists you can jump to viewpoints, and operate on the paths.

For paths, you can

- Play
- Suspend
- Stop

them. When you play a path, the DeskTop will disable all user interaction and play the path, until it is stopped or paused. Stopping the path will terminate the path and restore the User Interface of the DeskTop to whatever it was before the path started. Pausing the path will stop the path playing but keep it active ready to continue. Issuing another pause command will restart the path.

Users of the API cannot create viewpoints or paths, they can only replay them.

Functions and Events

Function Name	Description
SelectViewpoint	Tells the Desktop to jump to a viewpoint.
SelectPath	Tells the DeskTop to play, pause or stop a path.
SendViewpoints	Instruct the DeskTop to send a list of viewpoints in the current model.
SendPaths	The DeskTop will send a list of all the paths known in the current model.

Event Name	Triggered By
OnViewpoints	DeskTop receives a **SendViewpoints** command.

Event Name	Triggered By
OnPaths	DeskTop receives a **SendPaths** command.

HRESULT SelectViewpoint(BSTR name);

Parameters

name = the name of the viewpoint you want to jump to.

Description

Instructs the DeskTop to jump to the viewpoint with the value specified by **name**. If there is no viewpoint with that value then the instruction is ignored. Use **SendViewpoints** to get a list of all viewpoints within a model.

The **name** value must be an exact match, and is case sensitive.

HRESULT SelectPath(ViPathAction action, BSTR name);

Parameters

action = what you want to do to the path

name = the name of the path to use

Description

Instructs the DeskTop to play, pause or stop a path.

Use **SendPaths** to send a list of all the paths in the current model. The **name** value must match exactly with one in the model. Case is significant. If the path does not exist then the instruction is ignored. Action can be one of:

- **ViPlay**(=2) - Play the path starting from the first frame, and switch off all user interface elements except the path toolbar.
- **ViPause** (=1) - Stop playing the path but do not restore the user interface. Calling this twice will continue playing the path from the point it was paused at.
- **ViStop** (=0) - Stop playing the path and restore the user interface.

HRESULT SendViewpoints();

Parameters

None

Description

Instruct the DeskTop to send a list of viewpoints in the current model. This will cause a **OnViewpoints** event to be triggered.

HRESULT SendPaths();

Parameters

None

Description

Issuing this call will cause an **OnPaths** event to be triggered. The DeskTop will send a list of all the paths known in the current model.

HRESULT OnViewpoints(VARIANT names);

Parameters

names = array of strings

Description

Event that is triggered when the DeskTop receives a **SendViewpoints**command. **names** contains a list of the names of all viewpoints known in the current model at the time the SendViewpoints command was received.

HRESULT OnPaths(VARIANT names);

Parameters

names = array of strings

Description

Event that is triggered when the DeskTop receives a **SendPaths**command. **names** contains a list of the names of all paths known in the current model at the time the **SendPaths** command was received.

3

Invoking the API

This chapter covers the following topics:

- Visual Basic
- Visual C++
- HTML (VBScript)

Visual Basic

Inside Visual Basic you need to do the following steps to be able to use the API within a project.

1. From the **Project** menu, choose **Components**.
2. Choose the **SynergyAPI** type library from the list.
3. Click **OK**.
4. An icon will be displayed on your toolbar. This is the **Synergy** object.
5. Drag that icon onto your dialog.
6. You will now be able to execute Synergy API commands.

Visual C++

To create a Visual C++ project in which you can use the Synergy API, follow the steps listed below.

1. Create a new workspace.
2. Add a new mfc project to the workspace and make it dialog based (in order to see how to access Synergy API in fewer steps).

3. Make sure that you enable automation. Other than that, use the default settings.

4. Open the resource editor and right mouse click on the dialog.

   Choose **Insert ActiveX object**.

   Pick the **Synergy** class from the list that pops up. (The object itself has no user interface so it will have no visual affect on the contents of the dialog.)

5. Go to the class wizard and you will see something like **IDC_SYNERGY1**.

   Click on it and you will see all the events provided by the Synergy API.

   Select the event you want and press **Add Function**. Then you can fill out the body as you would for any other mfc callback.

6. Go to the **Member Variables** tab in the class wizard.

   Click on **IDC_SYNERGY1** and choose **Add Variable**.

   You will see something about creating a header file from the object. Just click on **OK** and give the object an appropriate name, e.g., **m_Synergy**.

HTML (VBScript)

In order to use the Synergy API in HTML pages, you must use VBScript within Internet Explorer.

To insert the API into an HTML page use the following code:

```
OBJECT classid=clsid:C2F3AE8D-7738-11D3-A3E4-0800690F45C8
id=synergy></OBJECT>
<SCRIPT ID=clientEventHandlersVBS LANGUAGE=vbscript>
```

This will define the Synergy object and setup the page to use VBScript by default.

> **Note:** VBScript does **not** support enumerated types. For function parameters that use enumerated types, you must specify the value directly. For enumerated types, the values are provided in parentheses for your convenience.

A

File Locations

This appendix covers the following topics:

- File Locations
- Manual Installation
- The Shareddesktop.exe Application

File Locations

The installer should install everything with no user input required. The details here are just in case you wish to verify that everything installed properly.

The API consists of three files:

- SynergyAPI.dll
- SharedDesktopps.dll
- SharedDesktop.exe

Manual Installation

Installation consists of the performing the following commands from the directory that contains the DLLs and executable:

```
regsvr32 SynergyAPI.dll
regsrv32 SharedDesktopps.dll
SharedDesktop /RegServer
```

The Shareddesktop.exe Application

SharedDesktop.exe is started automatically whenever anyone uses the Synergy API. It is a very small application that manages the communication between DeskTop and

others. It should automatically terminate when no one is using the API.

In some cases it may not terminate properly. If this happens, it does no harm to leave it running. It will be used by the next application to use the API. The only downside of this is that the value in **NumberOfConnections** may not be correct. **Shareddesktop** will discard unused connections after 5 minutes of no communication, so **NumberOfConnections** will converge to the correct value eventually.

B

Code Samples

This appendix covers the following topics:

- Code Samples

Code Samples

Code samples are provided to demonstrate the use of the API. The electronic files for these examples can be found in the directory:

```
C:\Orant\CADView-3D\SynergyAPI\Samples
```

Subdirectories are provided for Visual Basic and Visual C++.

Look at the following files:

- Alive
- Annotations
- Collaboration
- CollabVote
- PartsViewer
- Selection
- UserInterface

for examples of how to use the different aspects of Synergy API.

C

Quick Reference to API Functions, Events and Properties

This appendix covers the following topics:

- API Functions
- API Events
- API Properties

API Functions

Function Name	Description
ControlEnvisioni	Modify CADView-3D's run time behavior.
DisableUserModification	Control whether the CADView-3D user can change which toolbars are on or off.
Disconnect	Inform the Synergy API to close all connections. Should be the last operation performed.
ExtractAnnotationImage	Extract the image of the annotation and save onto the disk.
Flush	Empty any queued messages. If you want to queue up messages make sure that you execute this first.
GetStreamInfo	Gets stream information.

Function Name	Description
InsertAnnotation	Create a new annotation with the image pointed to, using the URL and descriptive text supplied.
SelectAction	Select the action to perform (highlight, fit, hide, show, etc.) on the current selection.
SelectAll	Select all shapes in the stream or all streams.
SelectById	Select the shapes with the specified IDs.
SelectByIdRange	Select a range of shapes by using the start and end IDs for the range of objects.
SelectByName	Select the named object.
SelectByQuery	Select objects by performing a database query
SelectBySQL	Select the object matching the attribute found by the specified SQL query.
SelectClearAll	Clear all objects currently selected.
SelectPath	Stop/pause/play the specified path.
SelectSpatially	Select objects within a sphere.
SelectViewpoint	Jump to the named viewpoint in the current model.
SendAttributes	Ask the DeskTop to send all attributes for an object.
SendAttributeTypes	Ask for a list of attribute types to be sent via an OnAttributeTypes event.
SendCollaborationMessage	Send a message the collaboration group. The user is entirely responsible for what this data means.

Function Name	Description
SendNames	Ask the DeskTop to send the names of objects in a range.
SendPaths	Ask CADView-3D to send a list of known paths.
SendSelected	Return information about current selection.
SendViewpoints	Ask CADView-3D to send a list of all known viewpoints.
SetHighlightMode	Sets what the affect the highlight action will have.
SetMovementMode	Switch between drive and examine modes.
SetOrientation	Set the view orientation in degrees.
SetPosition	Set the XYZ position.
StartEnvisioni	Start CADView-3D in the default browser with the URL given.
ToolbarVisibility	Switch selected toolbar on or off.

API Events

Event Name	Occurs when
OnAnnotationReturn	An annotation extraction has been requested and completed.
OnAttributes	A SendAttributes request is made.
OnAttributeTypes	A SendAttributes request is made.
OnClientAlive	A DeskTop comes alive.

Event Name	Occurs when
OnClientDeath	A DeskTop shuts down.
OnCollaborationMessage	Application has sent a collaboration message. Used in conjunction with OnCollaborationState.
OnCollaborationStateChange	Collaboration state changes.
OnErrorOccurred	An error occurs.
OnNames	A SendNames request is made.
OnOrientation	User changes orientation inside CADView-3D.
OnPaths	A SendPaths command is made.
OnPick	User performs a pick.
OnPosition	The view position changes.
OnSelectedCount	A SendSelected request is made.
OnSelectedData	A SendSelected request is made.
OnSelectedIds	A SendSelected request is made.
OnViewpoints	A SendViewpoints command is made.

API Properties

Property Name	Type	Usage	Description
Error Coce	Short	Read only	Contains the last error code.
ErrorString	BSTR	Read only	Contains text of the last error.

Property Name	Type	Usage	Description
NumberOfConnections	Long	Read only	Number of connections to the Synergy API.
NumberOfObjects	Long	Read only	Total number of objects in the database.
NumberOfStreams	Short	Read only	The number of streams that the DeskTop is using.
PollRate	Long	Read/Write	Time in milliseconds between Synergy API checks for events. (Discussed in detail under Property PollRate.)
SourceAm3File	BSTR	Read only	The .AM3 file currently in use.

Index

P

R

S

T

V