Oasys

Oasys GSA

API Reference

Oasys
YOUR IDEAS BROUGHT TO LIFE

13 Fitzroy Street
London
W1T 4BQ
Telephone: +44 (0) 20 7755 3302
Facsimile: +44 (0) 20 7755 3720

Central Square
Forth Street
Newcastle Upon Tyne
NE1 3PL
Telephone: +44 (0) 191 238 7559
Facsimile: +44 (0) 191 238 7555

e-mail: oasys@arup.com
Website: oasys-software.com

Oasys GSA

Contents

Introduction

GSA APIs allow other programs and scripts to programmatically access to GSA functionality. The APIs are implemented using Microsoft's COM technology. They allow GSA models to be created, edited, analysed and also allow results to be queried.

GSA APIs can be invoked by any programming language or scripting environment that can work with COM and ActiveX. Examples include .NET (c# and vb), Python, VBA, C++ and MATLAB. We recommended the use of .NET to interact with GSA APIs – this environment is closely compatible with the APIs, and there are several code samples get your started. Python is also a good alternative.

All the API is exposed via the type library file Gsa.tlb, which is also wrapped as a .NET interop dll 'Interop.Gsa_9_0.dll', both of which are available in the GSA program files folder. The library exports a single object: `ComAuto` and several `structs` and `enums`. These are listed in the API reference section in the later part of this document.

Note that function names are case sensitive. A log file is created in the same directory as the GSA model file to record the execution of each of the functions.

The COM interface is 'versioned', i.e. each minor and major release of GSA has COM classes specific to that release. Instantiating these classes will specifically invoke the version of GSA they correspond to. Whilst this gives the flexibility to bind to a particular release of GSA, it is also possible to "always" bind to the latest release. The choice of whether a programmer wants to bind to a version specific COM class or version independent COM depends on how s/he invokes GSA COM (see the section Early and Late Binding).

Getting Started

The API can be thought of made up of "families of functions" for ease of understanding. This is reflected in the way it is documented in the reference.

A good way to start using the API is by using one of the several code samples as a starting point. The rest of this section demonstrates a basic sample in two languages: VB.NET and Python.

There are samples for .net, Excel/vba and C++ under the Samples\API_and_GWC folder under the GSA program files subdirectory. Each sample demonstrates the use of a section of the API. The file < C:\Program Files\Oasys\GSA 9.0\Samples\API_and_GWC\readme.txt > lists the samples and corresponding family of functions that is used by the sample.

The first step in using the APIs is to import the type library or the interop dll into your programming environment. The COM interface is 'versioned', i.e., each minor and major release of GSA has COM classes specific to that release. As an example, for using the GSA 9 API in vb.NET, import < C:\Program Files\Oasys\GSA 9.0\Interop.Gsa_9_0.dll> and then instantiate the object as follows:

```
Dim gsaObj As Gsa_9_0.ComAuto
```

```
Set gsaObj = New Gsa_9_0.ComAuto
```

Once we instantiate the objects, we can proceed to view the available functions in the 'Object browser' (if using Visual Studio) and call them.

This basic sample creates the object, opens the GSA file, deletes results and runs all analysis tasks in the file.

VB.NET

```
Dim gsa As New ComAuto

' Write out the version of GSA that has been invoked
Console.WriteLine(gsa.VersionString())

' Open GSA file. (Ensure that the file exists at this path)
gsa.Open("c:\temp\stair.gwb")

' delete any analysis results in the file
gsa.Delete("RESULTS")

' analyse all tasks
gsa.Analyse()

' save the analysed model with a different file name
gsa.SaveAs("c:\temp\stair_analysed.gwb")

gsa.Close()

gsa = Nothing
```

Python

```
# For a description of each line, see comments in the vb.net sample
import win32com.client
import pythoncom
gsaobj = win32com.client.Dispatch("Gsa_9_0.ComAuto")
gsaobj.VersionString()
gsaobj.Open("C:\\temp\\Stair.gwb")
gsaobj.Delete("RESULTS")
gsaobj.Analyse()
gsaobj.SaveAs("c:\\temp\\stair_analysed.gwb")
gsaobj.Close()

gsaobj = None
```

API function reference

GSA COM API functions are grouped as families of functions. This classification is purely for ease documentation and understanding – in reality all the functions below are offered by the `ComAuto` object. The families are:

- Core functions
- View functions
- GwaCommand function
- Data functions
- Output functions
- Case and Task functions
- List functions
- Tool functions
- Utility functions

Core functions

integer NewFile ()

Open a new model.

Returns status

0 – OK

1 – failed to open

integer Open (filename)

Open a GWB, GWA or CSV file.

Returns status

0 – OK

1 – failed to open

Argument

filename – the name of the file to be opened, including path and extension.

integer Save ()

Save the data to the default file (i.e. overwriting the file that was opened or last saved).

Returns status

0 – OK

1 – no GSA file is open

2 – no default path is available; use SaveAs

3 – failed to save

integer SaveAs (filename)

Save the data to GWB, GWA or CSV file.

Returns status

0 – OK

1 – no GSA file is open

2 – invalid file extension

3 – failed to save

Arguments

filename – the name of the file to be saved, including path and extension.

integer Close ()

Close the current file.

Returns status

0 – OK

1 – no GSA file is open

integer Analyse (integer task)

Analyse the tasks in GSA.

Where the task is greater than zero then analyse that task. If the task is zero or less, and no analysis tasks exist then do linear static analysis of each specified load case. If the task is zero or less, and the analysis tasks exist then analyse all analysis tasks that have not been analysed.

Returns status

0 – OK, analysis attempted

(use CaseResultsExist or TaskStatus to establish whether the analysis succeeded in producing results)

1 – no GSA file is open

2 – failed to attempt analysis

Arguments

task – analysis task number. Set to -1 (default) to run all analysis tasks.

integer Delete (string option)

Deletes results.

Returns status

0 – OK

1 – no GSA file is open

2 – invalid option

3 –data is not present; – no action taken

Arguments

option – valid settings are:

RESULTS – delete all results but not analysis cases.

RESULTS_AND_CASES – delete all results and analysis cases.

integer Design(integer task, enum DesignOption option)

Runs the specified design task.

Returns status

1 – No GSA file open

2 – Design task does not exist

3 – Design task has results.

Arguments

task – the reference of the Design task to be executed. A value of 0 indicates all tasks.

option – enum of the type DesignOption that specifies if the task will run as a 'design' or as a 'check'.

View functions

UpdateViews ()

Refreshes all GSA views currently displayed.

integer PrintView (string option)

Print saved or preferred Graphic Views or Output Views. Returns a status, as follows:

0 – OK

1 – no GSA file is open

2 – invalid argument

Arguments

option – valid settings are:

ALL_PGV – print all preferred Graphic Views.

ALL_SGV – print all saved Graphic Views.

ALL_POV – print all preferred Output Views.

ALL_SOV – print all saved Output Views.

ALL_LST – print all View Lists.

TAGGED_PGV – print tagged preferred Graphic Views.

TAGGED_SGV – print tagged saved Graphic Views.

TAGGED_POV – print tagged preferred Output Views.

TAGGED_SOV – print tagged saved Output Views.

TAGGED_LST – print tagged View Lists.

<name> – print the first view or view list found with the specified name.

integer SaveViewToFile (string option,string filetype)

Save saved or preferred Graphic Views or Output Views to file. Returns a status, as follows:

0 – OK

1 – no GSA file is open

2 – invalid argument

Arguments

option – valid settings are:

ALL_PGV – save all preferred Graphic Views.
ALL_SGV – save all saved Graphic Views.
ALL_POV – save all preferred Output Views.
ALL_SOV – save all saved Output Views.
ALL_LST – save all View Lists.
TAGGED_PGV – save tagged preferred Graphic Views.
TAGGED_SGV – save tagged saved Graphic Views.

TAGGED_POV – save tagged preferred Output Views.
TAGGED_SOV – save tagged saved Output Views.
TAGGED_LST – save tagged View Lists.
<name> – save the first view or view list found with the specified name.

filetype – valid settings are:

WMF – save Graphic Views to WMF file.
PNG – save Graphic Views to PNG file.
JPG – save Graphic Views to Jpeg file.
TXT – save Output View to tab delimited TXT file.
CSV – save Output View to comma delimited CSV file.
HTM – save Output View to HTML file.

integer HighestView (string option)

Return the highest numbered saved or preferred Graphic View or Output View.

option – valid settings are:

PGV – highest numbered preferred Graphic View.
SGV – highest numbered saved Graphic View.
POV – highest numbered preferred Output View.
SOV – highest numbered saved Output View.
LST – highest numbered saved View List.

integer ViewExist (string option, integer ref)

Returns 1 if the saved or preferred Graphic View or Output View exists.

option – valid settings are:

PGV – preferred Graphic View.
SGV – saved Graphic View.
POV – preferred Output View.
SOV – saved Output View.
LST – saved View List.

ref – view number (1 based)

string ViewName (string option, string ref)

Returns the name of the saved or preferred Graphic View or Output View.

Arguments

option – valid settings are:

PGV – preferred Graphic View.
SGV – saved Graphic View.
POV – preferred Output View.
SOV – saved Output View.
LST – saved View List.

ref – view number (1 based)

integer ViewRefFromName (string option, string name)

Returns the saved graphic view or output view reference from its name.

option – valid values are:

SGV – saved Ouput View
SOV – saved Output View

name – the name of the saved view.

integer CreateNewView(string name)

Creates a new saved graphic view and returns its index.

name – the intended name of the created graphic view.

integer SetViewBaseSettings(integer id, string SavedViewGwa)

Applies "base" view settings from the saved view template provided onto the given saved view referenced by id.

Returns status

1 – no file is open

2 – Saved graphic view with id does not exist

3 – SavedViewGwa string is empty or invalid

4 – could not read a valid saved view from SavedViewGwa

0 - OK

Arguments

id – the index of the saved view to which you want to apply the template.

SavedViewGwa – the GWA record of a saved view that is to be used as a template for the contouring settings that you want to apply to the above view.

Note

See the documentation to SetViewContour for an explanation of the template and for the general use/workflow of this function.

integer SetViewContour(integer id, integer dataref, string SavedViewGwa)

Applies the contour settings from the saved view template provided onto a given saved view referenced by id. Then sets the output component that is contoured to 'dataref'.

Returns status

1 – no file is open

2 – Saved graphic view with id does not exist

3 – SavedViewGwa string is empty or invalid

4 – could not read a valid saved view from SavedViewGwa

0 - OK

Arguments

id – the index of the saved view to which you want to apply the template.

dataref – the dataref of the result component that you want to contour. See 'Output_Init' for an explanation of how datarefs work.

SavedViewGwa – the GWA record of a saved view that is to be used as a template for the contouring settings that you want to apply to the above view.

Note

The SetViewContour, SetViewDiagram, SetViewLabels, and SetViewBaseSettings functions allow settings to be applied based on previously created saved views that are used as templates. The templates do not need to be in the same GSA file – all that's needed is the GWA record of the template. These could be generated in one of the two ways:

- Create a saved view with all the settings you would like to see in your final view and save the file as GWA. Open in text editor, and copy the relevant GR_VIEW record and paste in your code as a string constant.
- Alternatively, create and save a GSA model with view templates as above. At runtime, use your program to open this GSA file, and get the GWA record of the appropriate template using the GwaCommand function.

The following VB.NET code demonstrates the above workflow.

```
        Dim gsa As New ComAuto
        ' this is the GSA file that has our view templates
        gsa.Open("viewtemplates.gwb")

        Dim ref_template_contour = gsa.ViewRefFromName("SGV", "template-contour")
        Dim ref_template_labels = gsa.ViewRefFromName("SGV", "template-labels")

        Dim contour_template As String = gsa.GwaCommand("GET, GR_VIEW," &
ref_template_contour)
        Dim label_template As String = gsa.GwaCommand("GET, GR_VIEW," &
ref_template_labels)
        gsa.Close()

        gsa.Open("steel_design_medium.gwb")

        ' Create a new saved view that will display a contour
        Dim viewRef As Integer = gsa.CreateNewView("contour")

        ' Now "apply" contour_template to this view and set
        ' the result component to nodal displacement, z
        gsa.SetViewContour(viewRef, 12001003, contour_template)

        ' Turn labels on by apply the labels template
        gsa.SetViewLabels(viewRef, label_template)
```

integer SetViewLabels(integer id, string SavedViewGwa)

Applies labels settings from the supplied view template onto the saved view with the given reference.

Returns status

1 – no file is open

2 – Saved graphic view with id does not exist

3 – SavedViewGwa string is empty or invalid

4 – could not read a valid saved view from SavedViewGwa

0 - OK

Arguments

id – the index of the saved view to which you want to apply the template.

SavedViewGwa – the GWA record of a saved view that is to be used as a template for the contouring settings that you want to apply to the above view.

Note

See the documentation to SetViewContour for an explanation of the template and for the general use/workflow of this function.

integer SetViewDiagram(integer id, integer dataref, string SavedViewGwa)

Applies the diagram settings from the saved view template provided onto a given saved view referenced by id. Then draws the diagram for the output component represented by 'dataref'.

Returns status

1 – no file is open

2 – Saved graphic view with id does not exist

3 – SavedViewGwa string is empty or invalid

4 – could not read a valid saved view from SavedViewGwa

0 - OK

Arguments

id – the index of the saved view to which you want to apply the template.

dataref – the dataref of the result component that you want to contour. See 'Output_Init' for an explanation of how datarefs work.

SavedViewGwa – the GWA record of a saved view that is to be used as a template for the contouring settings that you want to apply to the above view.

Note

See the documentation to SetViewContour for an explanation of the template and for the general use/workflow of this function.

integer GetViewOrientation(integer id, ref GsaViewOrientation orientation)

Returns the view orientation settings of a saved graphic view in the form of a GsaViewOrientation object (see struct GsaViewOrientation).

Returns status

1 – no file is open

2 – Saved graphic view with id does not exist

Arguments

id – the index of the saved view to which you want to apply the template.

orientation – An object of struct GsaViewOrientation.

The GsaViewOrientation struct contains fields that correspond to the settings from the Graphics settings > Orientation dialog in an interactive session.

integer SetViewOrientation(integer id, GsaViewOrientation orientation)

Sets the view orientation settings of a saved graphic view as supplied in the GsaViewOrientation object (see struct GsaViewOrientation).

Returns status

1 – no file is open

2 – Saved graphic view with id does not exist

3 – Error in the definition of orientation object: one of `orientation` or `MidPoint` or `ObjectPoint` are null.

0 - OK

Arguments

id – the index of the saved view to which you want to apply the template.

orientation – An object of struct GsaViewOrientation.

The GsaViewOrientation struct contains fields that correspond to the settings from the Graphics settings > Orientation dialog in an interactive session.

integer RescaleViewToFit(integer id)

Rescales the saved graphic view associated with the supplied id.

Returns status

1- no file is open

2- Saved graphic view with id does not exist

Arguments

id - the index of the saved view that you want to rescale.

Note

This function has the same effect as the 'Scale to fit' command in an interactive session.

integer RescaleViewData(integer id)

Rescales the contour and diagram data (i.e. recalculates extents, etc.) for the graphic view associated with the supplied id.

Returns status

1- no file is open

2- Saved graphic view with id does not exist

Arguments

id - the index of the saved view that you want to rescale.

Note

This function has the same effect as the 'Rescale data' command in an interactive session.

GwaCommand function

GWA commands are commands that interact directly with the data in a GSA session. This allows the user to modify any of the data in a GSA file. GSA data is stored in modules that are one of the following types:

- specification – single record module (e.g. General Specification)
- ordered table modules – multi-record but can include gaps in the records (e.g. Nodes)
- collection table modules – multi-record without gaps in the records (e.g. Beam Loads)

Generally collection table modules are those that are not cross referenced with other modules, so, for example, switching a couple of load records does not affect the data integrity while switching a couple of node records does.

The syntax of the command is based on GWA syntax and the units follow the GWA unit syntax; – refer to the "GSA Text (ASCII) File" chapter for details.

variant GwaCommand (string command)

Issue a GWA command to GSA.

Argument

command – a command in GWA format with fields separated by a tab or the list separator (typically a comma or semi-colon). The commands that can be used are:

key_word, {data} – this is used to insert a record of data (e.g. writing a complete set of data to a GSA model).

SET, key_word, {data} – this is similar to the previous command and inserts a record of data (for specification and ordered table modules). For a node this will overwrite the node (if it is already defined) as the node number is part of the data – but for a load it will simply append to the module as the load records do not have a record number).

SET_AT, record, key_word, {data} – set a particular data record (for collection table modules) this allows, for example, load records to be modified and overwritten as the record number is specified.

ADD, key_word, {data} – add a data record at the end of the module (for collection table modules).

GET, key_word, {data} – return a record of data.

HIGHEST, key_word – return the number of the highest record number.

BLANK, key_word, low_record [, high_record] – blank a record or a range of records (without reordering subsequent records).

DELETE, key_word, low_record [, high_record] – delete a record or a range of records (reordering subsequent records to close the gap).

EXIST, key_word, record – return true if the record exists.

LOCKED, key_word, record – return true if the record is locked.

– where items in [ ] are optional and items in { } represent a list of values.

Examples of ***command***:

To create a node 25 at (10,4.5,8)

SET, NODE, 25, 10, 4.5, 8

or

UNIT_DATA, LENGTH , cm, 100

NODE, 25, 1000, 450, 800

To set beam load 1 to be a UDL of −1000 in z on elements 1 to 10 for case10 in global axes

SET, 1, LOAD_BEAM, 1 to 10, 10, GLOBAL, NO, Z, -1000

or to append this beam load

ADD, , LOAD_BEAM, 1 to 10, 10, GLOBAL, NO, Z, -1000

To check if element 11 exists

EXIST, EL, 11

To create a beam element 11 with property 1, group 101 and nodes 12 and 22 and orientation node 3

SET, EL_BEAM, 11, 1, 101, 12, 22, 3

To return the data for material 3

GET, MAT, 3

To get the displacements for node 25 in case 4

GET, DISP, 25, 4

To check the highest node

HIGHEST, NODE

To see if an axis 4 is locked

LOCKED, AXIS, 4

To blank element 40

BLANK, EL, 40

or to delete the element

DELETE, EL, 40

Notes on using GwaCommand:

- The GwaCommand fully exposes the GSA data structure, so care should be taken to ensure that the data is not corrupted.
- When creating **elements** the entity type is specified in the keyword, so to create a beam element the keyword is EL_BEAM but when inquiring about an element the keyword is simply EL.
- To check the highest **user module** or to delete a user module refer to the respective user module title record. E.g.

  HIGHEST, USER_MOD_NODE_TITLE

  DELETE, USER_MOD_ELEM_TITLE, 3

String IDs (sIDs) and GwaCommand:

- The model sID is accessed by using the SID key_word.

  SET, SID, {tag1:value1}{tag2:value2}

- Object sIDs are appended to the object key_word.

SET, EL_BEAM:{tag1:value1}{tag2:value2}, 11, 1, 101, 12, 22, 3

Refer to Working with the Program — String IDs for information on sID formatting.

Data functions

integer NodeCoor(integer id, double x, double y, double z)

Retrieves the position of a node in global coordinates.

Returns status

0 – OK

1 – node not found

Arguments

ref – node reference

x – x coordinate of the node (double*)

y – y coordinate of the node (double*)

z – z coordinate of the node (double*)

integer Gen_NodeAt (double x, double y, double z, double tol)

Returns the node number of the newly generated node or the existing node for the given coordinate(x, y, z).

Arguments

x – x coordinate of the node

y – y coordinate of the node

z – z coordinate of the node

tol – tolerance to use existing node

string Gen_SectionMatchDesc (string sectdesc, integer flags)

Returns valid GSA section description from the given tentative section description (sectdesc)

sectdesc – tentative section description

flags – compound flag; valid settings are:

Enum Gen_SectionMatchDesc_Flags

SEC_INCL_SS = &H1 include superseded sections in search

SEC_ATTEMPT_STD = &H2	attempt to assemble a standard section from tentative section description (sectdesc)

End Enum

integer MembNumElem (integer id)

Returns the number of elements associated with the given member.

Argument

id – member number

integer MembElemNum (integer id, integer index)

Returns the element number for the given member and element index.

Arguments

id – member number

index – element index for the member (zero based)

integer ElemMembNum (integer id)

Returns the member number of associated with the given element.

Argument

ref – element number

double Tool_GetEntLength(integer id, GsaEntity type)

Returns the length of the entity.

Arguments

ref – entity reference

type – an enum variable of type GsaEntity

integer Gen_RegionMeshCheck (integer id, ref integer error, ref integer warning, ref string message)

Check the validity of the given region.

Returns status

0 – no errors or warnings

1 – some errors or warnings

Arguments

id – region reference number (if ref is zero, all regions are checked)

error – the number of errors generated in running this function

warning – the number of warnings generated in running this function

message – the message explaining the status of running this function

integer Gen_RegionMeshGen (integer id, ref integer error, ref integer warning, ref string message)

Generate meshes for the given region (ref).

Returns status

0 – at least one region meshed successfully

1 – no region meshed

Arguments

id – region reference number (if ref is zero, meshes are generated for all regions)

error – the number of errors in running this function

warning – the number of warnings in running this function

message – the message explaining the status of running this function

integer Gen_RegionMeshDel (integer id)

Delete the mesh for the given region.

Returns status

0 – mesh for at least one region deleted successfully

1 – no region mesh deleted

Arguments

id – region reference number (if ref is zero, all regions are deleted)

integer Nodes(array nodeRefs, ref array nodes)

Fetches an array of GsaNode objects given an array of node references.

Returns status

0 – OK

1 – No file open or invalid input

Arguments

nodeRefs – integer array of valid node references

nodes – array of GsaNode objects associated with the node references

Note

As an example the following VB.NET snippet retrieves all nodes in the GSA model

```
Dim nodeRefs() As Integer
s = gsaObj.EntitiesInList("all", GsaEntity.NODE, nodeRefs)
Debug.Assert(Not s.Equals(0) And Not (nodeRefs Is Nothing))
Dim nodes() As GsaNode
s = gsaObj.Nodes(nodeRefs, nodes)
```

integer SetNodes(array nodes, bool Overwrite)

Sets an array of GsaNodes into the GSA model data.

Returns status

0 – OK

1 – No file open or invalid input

Arguments

nodes – array of GsaNode objects.

Overwrite – Boolean flag to indicate if existing nodes, if any, are to be overwritten

integer Elements(array elemRefs, ref array elems)

Fetches an array of GsaElement objects given an array of element references.

Returns status

0 – OK

1 – No file open or invalid input

Arguments

elemRefs – integer array of valid element references

elems – array of GsaElement objects associated with the element references

Note

For an example, see documentation for the function `Nodes(...)`.

integer SetElements(array elems, bool Overwrite)

Sets an array of GsaElement objects into the GSA model data.

Returns status

0 – OK

1 – No file open or invalid input

Arguments

nodes – SAFEARRAY of GsaElement objects.

Overwrite – Boolean flag to indicate if existing elements, if any, are to be overwritten

integer Sections(array sectRefs, ref array sections)

Fetches an array of GsaSection objects given an array of section references.

Returns status

0 – OK

1 – No file open or invalid input

Arguments

sectRefs –integer array of valid section references

sections –array of GsaSection objects associated with the section references

Note

For an example, see documentation for the function `Nodes(...)`.

integer SetSections(array sections, bool Overwrite)

Sets an array of GsaSection objects into the GSA model data.

Returns status

0 – OK

1 – No file open or invalid input

Arguments

sections – array of GsaSection objects.

Overwrite – Boolean flag to indicate if existing elements, if any, are to be overwritten

integer Members(array memberRefs, ref array Members)

Fetches an array of GsaMember objects given an array of member references.

Returns status

0 – OK

1 – No file open or invalid input

Arguments

sectRefs –integer array of valid member references

sections –array of GsaMember objects associated with the section references

Note

For an example, see documentation for the function `Nodes(...)`.

integer SetMember(array members, bool Overwrite)

Sets an array of GsaMember objects into the GSA model data.

Returns status

0 – OK

1 – No file open or invalid input

Arguments

members – array of GsaMember objects.

Overwrite – Boolean flag to indicate if existing elements, if any, are to be overwritten

integer NodeConnectedEnt(GsaEntity entityType, integer nodeRef, ref array entRefs)

Fetches the indices of elements or members connected to a node.

Returns status

0 – OK

1 – No file open or invalid input

2 – Node does not exist

Arguments

entityType – enum of type GsaEntity. Has to be ELEMENT or MEMBER.

nodeRef – reference of the node to query for

entRefs – array holding the references to connected entities. (Output)

Output functions

The following functions are for extracting processed data, for example, derived enveloped stresses.

integer Output_Init (integer flags, string axis, string case, integer dataref, integer num1dpos)

Initialises the output functions. Call this before calling any other "Output_" functions.

Returns status

0 – OK

1 – no GSA file is open

3 – invalid axis

4 – invalid case

5 – invalid dataref

Arguments

flags – compound flag; valid settings are:

Enum Output_Init_Flags

OP_INIT_2D_BOTTOM = &H1	output 2D stresses at bottom layer
OP_INIT_2D_MIDDLE = &H2	output 2D stresses at middle layer
OP_INIT_2D_TOP = &H4	output 2D stresses at top layer
OP_INIT_2D_BENDING = &H8	output 2D stresses at bending layer
OP_INIT_2D_AVGE = &H10	average 2D element stresses at nodes
OP_INIT_1D_AUTO_PTS = &H20	calculate 1D results at interesting points

End Enum

E.g. OP_INIT_2D_TOP Or OP_INIT_2D_AVGE 2D stresses at top layer, averaged at nodes

axis – output axis; enter the name of a standard axis or the number of a user defined axis; examples of valid entries:

"default" – the default for the data being extracted

"global"

"local"

case – the output case, ignored if not relevant; CasePermString may be used to collate this string; examples of valid entries:

"L1"

"A3"

"C3"

"C4max" – assumes C4 is an envelope

"C4min" – <ditto>

"C4abs" – <ditto>

"C4signabs" – <ditto>

"C4p3" – <ditto>

dataref – an integer data reference; refer to file “Output_DataRef.txt” in the Docs folder for available options; for example:

14003001 refers to beam element axial stress

num1dpos – the number of equidistant internal positions ainteger 1D elements to be considered for 1D element results, in addition to the automatic interesting positions if specified in flags

Note

The `dataref` field is determined by looking at the Output_DataRef file. First locate the result header that you are interested (for e.g. for Nodal displacement in y direction the header is `REF_DISP`). Then add the offset of the position of the result you seek. In the case of nodal displacement in y this corresponds to `REF_DISP_DY`, which is at an offset of 2 from `REF_DISP`, therefore the right value of dataref for `REF_DISP_DY` is 12001000+2 = 12001002.

The 'Output' worksheet in the < C:\Program Files\Oasys\GSA 9.0\Samples\API_and_GWC\vba.zip\vba\GsaComGwaSample.xls> file demonstrates the use of the Output family of functions. In particular, it can also been used to validate if you are working with the right value of the `dataref` argument. To validate, enter appropriate values for the file name, node/element ids and the `dataref`. Then click the Output button. The Title field should get populated with the name of the output quantity that `dataref` corresponds to.

integer Output_SetStage (integer stage)

Returns status

0 – OK

1 – no GSA file is open

2 – Output_Init not called

Argument

stage – the analysis stage to be considered; only relevant for input data (e.g. properties) since results are for the stage that was analysed; use 0 for "whole model"

string Output_DataTitle (integer flags)

Returns the title of the data specified in the last call to Output_Init, as a string.

Argument

flags – compound flag; valid settings are:

1 – full title (otherwise the abbreviated title is returned)

integer Output_IsDataRef (integer flags)

Returns 1 if the flags condition applies to the data specified in the last call to Output_Init.

Argument

flags – compound flag; valid settings are:

Enum Output_IsDataRef_Flags

```
    OP_IS_AND = &H1                 otherwise OR
    OP_IS_PER_REC = &H2
    OP_IS_PER_NODE = &H4
    OP_IS_PER_ELEM = &H8
    OP_IS_PER_MEMB = &H10
    OP_IS_PER_1D_DISP = &H20
    OP_IS_PER_1D_FRC = &H40
    OP_IS_PER_TOPO = &H80
    OP_IS_AT_CENTRE = &H100
End Enum
```

E.g. OP_IS_PER_1D_DISP Or OP_IS_PER_1D_FRC – data is reported at 1D element internal displacement OR internal force positions.

E.g. OP_IS_AND Or OP_IS_PER_ELEM Or OP_IS_PER_TOPO – data is reported per element AND per node per element.

string Output_UnitString ()

Returns the units of the data specified in the last call to Output_Init, as a string.

float Output_UnitFactor ()

Returns the factor to convert the data specified in the last call to Output_Init from SI to the current model units.

integer Output_DataExist (integer id)

Returns 1 if the data specified in the last call to Output_Init exists for the specified item.

Argument

id – the record or node or element or member number to be considered

integer Output_NumElemPos (integer id)

Returns the number of positions on the element or member for which the data specified in the last call to `Output_Init` are available. For 1D elements this will be the end positions plus the internal positions, based on the arguments supplied in the last call to `Output_Init`. For 2D elements this will be the number of nodal positions on the element plus, for some data options, the centre value.

Argument

ref – the element or member number to be considered

Note

Important: this function must be called before Output_1DElemPos and Output_Extract for 1D element results.

float Output_1DElemPos (integer pos)

Returns the position along a 1D element as a proportion of the element length for specified position number.

Argument

pos – the position number to be considered; zero based (i.e. “0” is the first position and “Output_NumElemPos − 1” is the last)

Note

Call `Output_NumElemPos` before calling this and after calling `Output_Init`.

variant Output_Extract (integer ref, integer pos)

Returns the requested data for the specified axis and case in the current model units. `Output_NumElemPos` should be called before calling this for 1D element data, and after calling `Output_Init`.

Argument

ref – the record or node or element or member number to be considered

pos – the position number to be considered; zero based (i.e. “0” is the first position and “Output_NumElemPos − 1” is the last)

integer Output_Extract_CurPerm ()

Returns the envelope permutation that gave the data returned by the last Output_Extract call. This is only relevant if the current case, set in Output_Init, is a "Cnmax", "Cnmin", "Cnabs" or "Cnsignabs" case.

integer Output_Init_Arr (integer flags, string axis, string case, enum ResHeader header, integer num1dpos)

Initializes the Output Array API for a specified case, axis, header and flags.

Returns status

0 – OK

1 – no GSA file is open

3 – invalid axis

4 – invalid case

5 – invalid dataref

Arguments

flags – compound flag; valid settings are:

```
Enum Output_Init_Flags

    OP_INIT_2D_BOTTOM = &H1              ' output 2D stresses at bottom layer
    OP_INIT_2D_MIDDLE = &H2              ' output 2D stresses at middle layer
    OP_INIT_2D_TOP = &H4                 ‘ output 2D stresses at top layer
    OP_INIT_2D_BENDING = &H8             ' output 2D stresses at bending layer
    OP_INIT_2D_AVGE = &H10               ' average 2D element stresses at nodes
    OP_INIT_1D_AUTO_PTS = &H20           ' calculate 1D results at interesting points
    OP_INIT_INFINITY = &H40              ' return infinity and NaN values as such,
    else as zero
    OP_INIT_1D_WALL_RES_SECONDARY = &H80     ' output secondary stick of wall
    equivalent beam results, else primary
End Enum

E.g. OP_INIT_2D_TOP Or OP_INIT_2D_AVGE ' 2D stresses at top layer, averaged at
```
nodes

axis – output axis; enter the name of a standard axis or the number of a user defined axis; examples of valid entries:

"default" – the default for the data being extracted

"global"

"local"

case – the output case, ignored if not relevant; `CasePermString` may be used to collate this string; examples of valid entries:

"L1"

"A3"

"C3"

"C4max" – assumes C4 is an envelope

"C4min" – <ditto>

"C4abs" – <ditto>

"C4signabs" – <ditto>

"C4p3" – <ditto>

header -- enum of the type, defined by ResHeader

num1dpos – the number of equidistant internal positions ainteger 1D elements to be considered for 1D element results, in addition to the automatic interesting positions if specified in flags

Note

The `Output_Init_Arr` and `Output_Extract_Arr` offer faster, array-based alternatives to the Output family of functions. The difference between the two is that array based functions work

with result headers, and therefore return all components of the requested result type in a single function call. For example, initializing `Output_Init_Arr` with `ResHeader.REF_DISP` returns all components of nodal displacements. When working with elements, the array based functions will retrieve all components along all points along an element.

integer Output_Extract_Arr(integer id, array results, integer numComponents)

Returns the output data for a node, element or member, as an array.

Arguments

id – the entity reference to return results for

results – array of struct GsaResults (Output)

numComponents – number of result components for the header in question (Output)

Note

The function `Output_Extract_Arr` returns an array of results for the element or node in question. For an element, the results array consists of an array of GsaResults objects. The number of objects returned is equal to one of the following:

- number of intermediate points on 1D elements
- topology points on 2D elements
- 1 if the entity in question is a node

Each `GsaResults` object has 2 members

- a dynamically allocated array called `dynaResults`. This contains double values corresponding to results for each component for the given header (as specified in the `Output_Init_Arr` function).
- an integer member NumComponents that contains the number of components in dynaResults.

For e.g. the call

```
gsaObj.Output_Set_Init(Output_Init_Flags.OP_INIT_1D_AUTO_PTS, "default", "A1",
ResHeader.REF_DISP, 2)
```

initializes the results API with REF_DISP, the nodal displacements.

From the file Output_DataRef.txt, we note that REF_DISP has the following components:

REF_DISP_DX, REF_DISP_DY, REF_DISP_DZ, REF_DISP_TRANS, REF_DISP_RXX, REF_DISP_RYY, REF_DISP_RZZ, REF_DISP_ROT, REF_DISP_DXY

Then, the call

```
gsaObj.Output_Extract_Arr(iNode, arrRes, numComp)
```

populates arrRes with a single struct that contains `dynaResults` of length 9 and value of `NumComponents` is set to 9.

The order of components in `dynaResults` is the same as their order in the `DataRef` enums.

integer Output_Extract_CutAssembly(integer ref, bool Avg2DStress, string case, string axis, array results)

Fetches 'Cut Section Forces' results for a given assembly. See documentation for the Cut Section Forces for more explanation of the parameters.

Arguments

ref – the assembly reference to extract forces for.

Avg2DStress – use averaged 2D stresses

case – case description

axis – axis definition

results – array of GsaResults object where

GsaResults.dynaResults stores the x, y, z xx, yy, zz results in that order,

GsaResults.Pos stores the position of the cut.

Note

For example:

```
Dim results() As GsaResults
gsa.Output_Extract_CutAssembly(1, False, 0, 0, "A1", "1", results)
```

Case and Task functions

The following functions are for extracting information on GSA load, analysis and combination cases.

integer HighestCase (string caseop)

Returns the highest numbered case.

Argument

caseop – the type of case; valid entries are:

"L" – load case

"A" – analysis case

"C" – combination case

integer CaseExist (string caseop, integer id)

Returns 1 if the case exists.

Argument

caseop – the type of case; valid entries are:

"L" – load case

"A" – analysis case

"C" – combination case

id – case number

string CaseName (string caseop, integer id)

Returns the name of the case as a string.

Argument

caseop – the type of case; valid entries are:

"L" – load case

"A" – analysis case

"C" – combination case

id – case number

integer CaseNumPerm (string caseop, integer id)

Returns the number of permutations in the case or 0 if the case is not an enveloping case.

Argument

caseop – the type of case; valid entries are:

"A" – analysis case

"C" – combination case

ref – case number

string CasePermDesc (string caseop, integer id, integer perm)

Returns the case description of the case as a string.

Argument

caseop – the type of case; valid entries are:

"A" – analysis case

"C" – combination case

id – case number

perm – permutation number or 0

string CasePermString (string caseop, integer id, integer perm)

Returns the case reference as a string, e.g. "C4p3".

Argument

caseop – the type of case; valid entries are:

"L" – load case

"A" – analysis case

"C" – combination case

id – case number

perm – permutation number or 0

float CasePermAnalFactor (string caseop, integer id, integer perm, integer analref)

Returns the factor of the specified analysis case that contributes to the specified case reference as a string, e.g. "C4p3".

Argument

caseop – the type of case; valid entries are:

"A" – analysis case

"C" – combination case

id – case number

perm – permutation number or 0

analref – the analysis case number; the returned factor of this analysis case contributes to the case specified by caseop, id and perm

integer CaseResultsExist (string caseop, integer id, integer perm)

Returns 1 if results exist for the case.

Argument

caseop – the type of case; valid entries are:

"A" – analysis case

"C" – combination case

ref – case number

perm – permutation number or 0

integer CaseTask (integer id)

Returns the reference number of the analysis task that is parent to the analysis case.

id – analysis case number

integer TaskStatus (integer id)

Returns the status of the analysis task.

Returns status

0 – task exists and has been analysed

1 – no GSA file is open

2 – task does not exist

3 – task exists but has not been analysed

Arguments

ref – analysis task number

integer DesignTaskStatus(integer id)

Returns the status of a design task.

Returns status

1 – no file open

2 – Design task does not exist

3 – Design task has results

4 – Design task has not been designed or checked.

Argument

id – design task reference.

List functions

integer IsItemIncluded (string option, integer id, string list)

Returns 1 is the item reference is included in the list

Arguments

option – valid settings are:

ITEM – a general list.
NODE – list is a list of nodes.
ELEM – list is a list of elements.

MEMBER – list is a list of members.
CASE – list is a list of cases.
GRID_PT – list is a list of grid points.

ref – item to be checked for inclusion

list – the list represented by a string (e.g. 1 3 5 or 1 to 10 not 7)

integer EntitiesInList (string lst, ref enum GsaEntity listType, array entities)

Retrieves entities from a list description or a saved list.

Returns status

0 – OK

1 – no GSA file open

2 – saved list does not exist

3 – no items in the list

Arguments

lst – list description or saved list reference (passed as a string)

listType – input or output parameter of type GsaEntity depending on whether `lst` is a list description or a saved list.

- If `lst` is a simple description (e.g. "1 to 10"), `listType` is an input parameter specifying the type of list
- If `lst` is a saved list reference (e.g. "1"), `listType` is an output parameter returning the type of the saved list

arrayEntities – the array of all valid entities in the list. (Output)

Tool Functions

integer Tool_UpdateElemSections ()

Returns true if successful

Call this function to update element section properties according to corresponding member's section properties.

integer Tool_ResetMemberSections ()

Returns true if successful

Call this function to reset member section properties according to associated elements' section properties.

Utility functions

short SetLocale (Locale locale)

Set the locale for Decimal and List Seperators for use by the GwaCommand function.

Arguments

option – enum of type Locale. Valid options are:

LOC_SYSTEM – Use the system default setting.

LOC_EN_GB – Use English.(',' and '.')

Returns status

0 – OK

1 – invalid Locale

The Locale setting is culture and language dependent and varies from one machine to the other. `GwaCommand` strings use decimal and list separators for input and output. Call this function to set `GwaCommand` to operator in a locale independent way.

integer NumArg(string line)

Returns the number of arguments in a line

Argument

line – a string of comma separated arguments

For example NODE, 25, 10, 4.5, 8 would return 5

string Arg(integer index, string line)

Returns as a string the argument at the index position in the line

Argument

index – index in line (zero based)

line – the line to be decoded

For example the third argument from NODE, 25, 10, 4.5, 8 would return 4.5

sID functions

Sid for each module record can be set and read though the `GwaCommand` using their format in GWA files. But this requires a lot of string parsing and hence the following functions are provided to set and get sids directly for each valid module record. For more information on sids, refer to String IDs

integer WriteSidTagValue(string key, integer record, string tag, string value)

Writes a Tag-Value pair to the sid of the module record specified.

Returns status

0 – OK, successfully written

1 – otherwise

Arguments

key – the keyword of the module

record – the index of the record

tag – the tag to be written to the sid

v**alue** – the value associated with the above tag

This function writes a Tag-Value pair if none exists inside the sid of the record. If there's already a tag present, the associated value is overwritten with the new value. Eg.(VB .NET)

```
Dim iSuccess as Integer = gsaObj.WriteSidTagValue("SEC_BEAM", 12, "RVT", "Structural
Framing - W44x285")
```

string GetSidTagValue(string key, integer record, string tag)

Returns the value for a tag stored inside the sid of the module record specified

Arguments

key – the keyword of the module

record – the index of the record

tag – the tag for which value is to be retrieved from the sid

An example of usage (VB .NET)

```
Dim sValue as String = gsaObj.GetSidTagValue("MEMBER", 3, "RVT")
```

Note

The `WriteSidTagValue` function sets the sid in the standard format specified in the documentation. Consequently the `GetSidTagValue` retrieves a value from sid only if it is in the same format.

Keyword for model sid is "SID".

An authoring application using the sid feature must write a model sid. Otherwise sids written to module records will not persist.

It's the authoring application's responsibility to ensure that the record exists for the module on which it is being called – otherwise GSA might throw an exception

Structs

These structs are used as arguments by various functions. The text in the brackets after each field provides an explanation of the field.

GsaViewOrientation

```
Latitude (Direction of view – Latitute)
Integeritute (Direction of view - Integeritude)
EyeDistance (Eye to object distance)
ObjectPoint (Object point coordinates as array of doubles)
MidPoint (Mid-point coordinates as array of doubles)
PictureRotation (Picture rotation)
```

GsaResults

```
NumComponents (Number of components available)
dynaResults() (Dynamic array of doubles)
Pos (Position of the cut -- applicable to Cut Section results only)
```

GsaElement

```
Ref (Reference)
Name
Color
eType (Element Type)
Property (Property reference)
Group (Group reference)
NumTopo (Number of topology nodes)
Topo() (Topology as an array of node references)
Beta (Orientation angle)
OrientNode (Orientation node)
Dummy (Dummy status)
```

GsaNode

```
Ref (Reference)
Name
Color
Coor() (Coordinates X, Y and Z as array of doubles)
Restraint (Nodal restraints as integer values)
Stiffness() (Stiffnesses in X, Y, Z, XX, YY, ZZ as array of doubles)
```

GsaSection

```
Ref (Reference)
Name
Color
SectDesc (Section Description string)
Material (Analysis Material)
eMatType (Material type – read-only)
MaterialGrade (Design Material)
Props() (Properties – read-only)
```

GsaMember

```
Ref (Reference)
Name
Color
Property (Property reference)
Group (Group)
Pool (Pool)
NumTopo (Number of topology nodes)
Topo() (Topology as an array of node references)
Beta (Orientation angle)
OrientNode (Orientation node)
MemberEnd_1 (Member End condition End 1)
MemberEnd_2 (Member End condition End 2)
```

Enums

```
enum GsaEntity// consistent with SelType

        NODE                        = 1,
        ELEMENT                     = 2,
        MEMBER                      = 3,
        LINE                        = 6,
        AREA                        = 7,
        REGION                      = 8



enum Locale

        LOC_SYSTEM          =       1,
        LOC_EN_GB           =       2



enum ResHeader  // consistent with Headers in

        REF_DISP                                        = 12001000,
        REF_VEL                                         = 12002000,
        REF_ACC                                         = 12003000,
        REF_REAC                                        = 12004000,
        REF_FORCE_CONSTR                            = 12005000,
        REF_FORCE_NODAL                                 = 12006000,
        REF_MASS_NODAL                                  = 12007000,
        REF_SOIL_NODAL                              = 12008000,
        REF_DISP_EL0D                               = 13001000,
        REF_FORCE_EL0D                              = 13002000,
        REF_DISP_EL1D                               = 14001000,
        REF_END_ROT_EL1D                            = 14001500,
        REF_FORCE_EL1D                                  = 14002000,
        REF_STRESS_EL1D                             = 14003000,
        REF_STRESS_EL1D_DRV                         = 14003200,
        REF_STRAIN_EL1D                             = 14003500,
        REF_SED_EL1D                                = 14004000,
        REF_SED_AVG_EL1D                            = 14005000,
        REF_STL_UTIL                                = 14006000,
        REF_DISP_EL2D                               = 15001000,
        REF_FORCE_EL2D_DRV                          = 15002000,
        REF_MOMENT_EL2D_PRJ                         = 15003000,
        REF_FORCE_EL2D_PRJ                          = 15004000,
        REF_STRESS_EL2D_DRV                         = 15005000,
        REF_STRESS_EL2D_AX                          = 15006000,
        REF_STRESS_EL2D_PRJ                         = 15007000,
        REF_RC_SLAB_REINF                           = 15010000,
        REF_DISP_EL3D                               = 15011000,
        REF_STRESS_EL3D_DRV                         = 15012000,
        REF_STRESS_EL3D_AX                          = 15013000,



ElementType   // consistent with ElementType

        BAR                 = 1,          // Bar (2 nodes)
```

```
        BEAM            = 2,            // Beam (2 nodes)
        SPRING          = 3,            // Spring (2 nodes)
        MASS            = 4,            // Mass element (1 node)
        QUAD4           = 5,            // Quad (4 nodes)
        QUAD8           = 6,            // Quad with mid-side nodes (8 nodes)
        TRI3            = 7,            // Triangle (3 nodes)
        TRI6            = 8,            // Triangle with mid-side nodes (6 nodes)
        LINK            = 9,            // Link (2 nodes)
        CABLE           = 10,           // Cable (2 nodes)
        BRICK8          = 12,           // 3d elements
        BRICK20         = 13,
        WEDGE6          = 14,
        WEDGE15         = 15,
        TETRA4          = 16,
        TETRA10         = 17,
        GRD_SPRING      = 18,           // Grounded spring (1 node)
        SPACER          = 19,           // Spacer (2 nodes)
        STRUT           = 20,           // Strut (2 nodes)
        TIE                     = 21,           // Tie (2 nodes)
        BEAM3           = 22,           // Curved Beam
        ROD                     = 23,           // Rod (2 nodes)
        DAMPER          = 24,           // Damper (2 nodes)
        GRD_DAMPER      = 25,           // Grounded damper (1 node)


enum RestraintDir    // consistent with Direction flags in gsdgen.h

        RD_FREE             = 0,    // Not restrained
        RD_X                = 0x1,  // Restrained for x force
        RD_Y                = 0x2,  // Restrained for y force
        RD_Z                = 0x4,  // Restrained for z force
        RD_PIN              = 0x7,  // Restrained for all force
        RD_XX               = 0x8,  // Restrained for x moment
        RD_YY               = 0x10, // Restrained for y moment
        RD_ZZ               = 0x20, // Restrained for z moment
        RD_RPIN             = 0x38, // Restrained for all moment
        RD_ENC              = 0x3f, // Restrained for force & moment
        RD_DIS              = 0x7,  // mask for force
        RD_ROT              = 0x38, // mask for moment
        RD_ALL              = 0x3f, // mask for all


enum DesignOption // consistent with DesignTask::Option

        CHECK = 0,
        DESIGN = 1,


enum MaterialType // consistent with Oasys::MatType

        UNDEF = 0,
        GENERIC = UNDEF,
        STEEL = 0x1,
        CONCRETE = 0x2,
        ALUMINIUM = 0x3,
        GLASS = 0x4,
        FRP = 0x5,
```

```
REBAR = 0x6,
TIMBER = 0x7,
FABRIC = 0x8,
SOIL = 0x9,
```

Other samples

Excel/VBA

The following is an example of invoking the GSA API from VBA script.

```
Sub RunGsa2()
    Dim iName0, iName1, iName2 As String
    Dim GsaAuto As Object
    Set GsaAuto = CreateObject("Gsa.ComAuto")
    iName0 = "C:\Temp\gsa_com0.gwa"
    iName1 = "C:\Temp\gsa_com1.gwa"
    iName2 = "C:\Temp\gsa_com2.gwb"
    GsaAuto.Open (iName0)
    GsaAuto.Analyse
    GsaAuto.Save
    GsaAuto.PrintView ("TAGGED_SGV")
    GsaAuto.Delete ("RESULTS")
    GsaAuto.SaveAs (iName1)
    GsaAuto.Delete ("RESULTS_AND_CASES")
    GsaAuto.SaveAs (iName2)
    GsaAuto.Close
    Set GsaAuto = Nothing
End Sub
```

COM C++ Example

The following is an example C++ code to run GSA remotely.

```
	if(FAILED(CoInitializeEx(0, COINIT_APARTMENTTHREADED)))
		return;

	IComAutoPtr pObj(__uuidof(ComAuto));
	short ret_code = 0;

	_bstr_t bsFileName = (LPCTSTR)filename;
	ret_code = pObj->Open(bsFileName);
	if(ret_code ==1)
		return;

	_bstr_t bsContent(_T("RESULTS"));
	ret_code = pObj->Delete(bsContent);
	ASSERT(ret_code != 1);	// file not open!

	_variant_t vCase(0L);
	ret_code = pObj->Analyse(vCase);
	ASSERT(ret_code ==0);
	_bstr_t bsAnalysedFileName = (LPCTSTR)analysed_filename;
	ret_code = pObj->SaveAs(bsAnalysedFileName);
	ASSERT(ret_code ==0);
	pObj->Close();
```

Early and Late Binding

In simplest terms, early and late binding refer to when COM programmers choose to let the compiler* know that the object created is of type Gsa.ComAuto.

In early binding, the object is declared to be of type Gsa.ComAuto and then invoked**.

```
Dim gsaObj As Gsa_8_4.ComAuto
Set gsaObj = New Gsa_8_4.ComAuto
```

In late binding, the object is simply declared to be of the type 'Object' and it's only when the code is run does the compiler know that it beintegers to the type Gsa.ComAuto.

```
Dim gsaObj As Object
Set gsaObj = CreateObject("Gsa.ComAuto")
```

When a particular approach is to be adopted over the other depends on particular needs(see reference 2) but broadly late binding allows for the code to be set up and prototyped quickly and is recommended if the programmer doesn't want to be bothered with the version of GSA he is using. Early binding on the other hand lets the code run faster and gives a better handle on the version of class being invoked.

References:

http://visualbasic.about.com/od/usingvbnet/a/earlybind.htm [about.com]

http://support.microsoft.com/kb/245115 [microsoft.com]

*Compiler/interpreter. In case of Excel, it's VBA.

** In VBA, this must be preceded by adding a reference to Gsa.tlb, from 'Tools | References'. Gsa.tlb can be found in the GSA program files folder. Other IDE's have equivalent methods.