



IST-2001-32133

GridLab - A Grid Application Toolkit and Testbed

## GAT API Specification

---

Author(s):	Tom Goodale
Document Filename:	Gridlab-1-GAS-0003.APISpecification
Work package:	Grid Application Toolkit
Partner(s):	PSNC,ZIB,MU,SZTAKI,VU,ISUFI,CARDIFF,GRIDWARE, COMPAQ,NTUA
Lead Partner:	MPG
Config ID:	GridLab-1-GAS-0003-0.1DRAFT
Document classification:	Internal

---

**Abstract:** This document provides a language independent description of the GAT API.



## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Purpose of Document	3
1.2	Structure of Document	3
1.3	Status of this Document	4
1.4	RFC 2119 and this Document	4
<b>2</b>	<b>API Function List</b>	<b>4</b>
2.1	GAT Application API	4
2.1.1	Job APIs	4
2.1.2	Resource APIs	4
2.1.3	Pipe APIs	4
2.1.4	File APIs	5
2.1.5	Monitoring APIs	6
2.1.6	Collection Management APIs	6
2.2	GAT Application Utility API	6
2.2.1	Management APIs	6
2.2.2	Asynchronous Event APIs	6
2.2.3	Error Handling and Auditing APIs	7
2.2.4	Security APIs	7
2.3	GAT Adaptor Registration API	7
2.4	GAT Adaptor Utility API	7
<b>3</b>	<b>GAT Application API Descriptions</b>	<b>7</b>
3.1	Job APIs	7
3.1.1	SubmitJob	7
3.1.2	KillJob	8
3.1.3	CheckpointJob	8
3.1.4	RestoreJob	9
3.1.5	MigrateJob	9
3.1.6	GetJobInfo	9
3.2	Resource APIs	10
3.2.1	FindResources	10
3.2.2	ReserveResources	10
3.2.3	UnReserveResources	10
3.3	Pipe APIs	11
3.3.1	CreatePipe	11
3.3.2	DestroyPipe	11
3.3.3	ListenOnPipe	11
3.3.4	ConnectPipe	12
3.3.5	SendToPipe	12
3.3.6	ReceiveFromPipe	12
3.4	File APIs	13
3.4.1	CopyFile	13
3.4.2	MoveFile	13
3.4.3	DeleteFile	13
3.4.4	OpenFile	14
3.4.5	CloseFile	14

3.4.6	ReadFromFile	14
3.4.7	WriteToFile	15
3.4.8	SeekInFile	15
3.4.9	PositionInFile	15
3.4.10	GetFileInfo	16
3.4.11	AddLocationOfFile	16
3.4.12	RemoveLocationOfFile	16
3.4.13	GetLocationsOfFile	17
3.4.14	ReplicateFileTo	17
3.5	Monitoring APIs	17
3.5.1	GetMetric	17
3.5.2	SubscribeToMetric	18
3.5.3	UnSubscribeFromMetric	18
3.6	Collection APIs	18
3.6.1	CreateCollection	18
3.6.2	DestroyCollection	19
3.6.3	AddToCollection	19
3.6.4	RemoveFromCollection	19
3.6.5	ListCollection	20
3.6.6	SetPropertiesOfCollection	20
3.6.7	GetPropertiesOfCollection	20
3.6.8	FindCollectionByProperties	21
3.6.9	FindObjectByProperties	21
<b>4</b>	<b>GAT Application Utility API</b>	<b>21</b>
4.1	Management APIs	21
4.1.1	Init	21
4.1.2	Destroy	22
4.1.3	CloneContext	22
4.2	Asynchronous Event APIs	22
4.2.1	ServiceEvents	22
4.3	Error Handling and Auditing APIs	23
4.3.1	GetErrorObject	23
4.3.2	DestroyErrorObject	23
4.3.3	SetAuditingLevel	23
4.4	Security APIs	24
4.4.1	CreateSecurityContext	24
4.4.2	DestroySecurityContext	24
4.4.3	GetSecurityContexts	24
4.4.4	Authenticate	25

## 1 Introduction

The GAT API provides an abstraction layer which abstracts operations which may need to be Grid-aware in such a way that an application may be developed and run independently of the underlying middle-ware actually deployed.

The GAT consists of a library – **the GAT Engine** – which all applications link against, and a set of **adaptors** which provide the bindings between the operations called in the application and the underlying providers, e.g. grid services. The technical reasons for this are outlined in the GAT Technical Specification [1].

The APIs can be split into four parts:

- GAT Application API

This is the API which provides the “grid” operations.

- GAT Application Utility API

These are GAT Engine management and interaction operations which an application needs to call to interact with the GAT.

- GAT Adaptor Registration API

This is the mirror of GAT Application API and is the way in which adaptors provide those functions.

- GAT Adaptor Utility API

These are the additional functions an adaptor needs to interact with the GAT Engine and possibly with other adaptors and utility libraries.

Since the GAT is an adaption layer we cannot enforce security at the API level, however we will provide functions which allow the application to specify credential information to the underlying adaptors which these may then use to authenticate to services and delegate, as determined by the underlying security model of these services.

### 1.1 Purpose of Document

This document provides a language-independent description of the GAT API. It is the first in a sequence of two language-independent API documents, and details the functions based upon the information they take in and return, and their affect. The second in the sequence will refine this document and define precise objects on which the functions operate, and assign attributes to these objects.

After acceptance of the second language-independent API specification, language-specific specifications will be developed. These will include adaptor-level APIs as well as the concrete-language bindings of the APIs described in the first two documents.

### 1.2 Structure of Document

This document consists of three sections. Section 2 lists all the API functions, section 3 details the operation of the GAT Application API, and 4 details the GAT Application Utility API.

There are no sections detailing the adaptor-layer APIs as these are language dependent.

### 1.3 Status of this Document

This document is still a draft. The API functions are still being worked on and the error states have not been enumerated. However it should be taken as a good indication of the set of API functions which will be available.

### 1.4 RFC 2119 and this Document

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [3].

## 2 API Function List

### 2.1 GAT Application API

While this document doesn't explicitly define objects, it was found to be advantageous while developing the API to think in terms of some generic classes of objects, in particular jobs, pipes, files, and resources. Additionally there was found to be a requirement to be able to group such objects into collections, and to advertise their presence.

#### 2.1.1 Job APIs

A job is a collection of tasks which are submitted together with one description to a resource management system, e.g. to a batch system on an individual resource or to the GridLab GRMS system [2].

- SubmitJob
- KillJob
- CheckpointJob
- RestoreJob
- MigrateJob
- GetJobInfo

#### 2.1.2 Resource APIs

- FindResources
- ReserveResources
- UnReserveResources

#### 2.1.3 Pipe APIs

Some applications, such as Triana [4], require peer-to-peer communications. For such applications we need to construct and manipulate bi-directional pipes through which data must be streamed. Note that at the implementation level such pipes may need to make use of sophisticated tunnelling and routing technologies to deal with firewalls and NAT.

- CreatePipe
- DestroyPipe
- ListenOnPipe
- ConnectPipe
- SendToPipe
- ReceiveFromPipe

**We may need a set of calls to do “select” on here, or perhaps that comes under monitoring so we can have application code triggered when some arbitrary event happens**

#### **2.1.4 File APIs**

Almost all applications need to access files, and need to have ways to be insulated from the real location of files.

- CopyFile
- MoveFile
- DeleteFile
- OpenFile
- CloseFile
- ReadFromFile
- WriteToFile
- SeekInFile
- PositionInFile
- GetFileInfo
- AddLocationOfFile
- RemoveLocationOfFile
- GetLocationsOfFile
- ReplicateFileTo

### 2.1.5 Monitoring APIs

Applications may wish to monitor the status of files, resources, jobs or even pipes.

- GetMetric
- SubscribeToMetric
- UnSubscribeFromMetric

Currently we have no APIs to allow an application to be monitored, but a monitoring adaptor can do this without the need of such an API. **Although perhaps we want an API call to switch this off and on and say who can monitor use.**

### 2.1.6 Collection Management APIs

Grouping of objects is a very common feature, which may be applied to files, pipes, jobs or even resources or information strings - in fact anything which may be advertised. This also serves as an advertisement of the existence of an object.

- CreateCollection
- DestroyCollection
- AddToCollection
- RemoveFromCollection
- ListCollection
- SetPropertyOfCollection
- GetPropertyOfCollection
- FindCollectionByProperties
- FindObjectByProperties

## 2.2 GAT Application Utility API

Not only must the API provide a set of calls for abstract grid operations, but an implementation must provide calls to initialise the library and do management tasks.

### 2.2.1 Management APIs

- Init
- Destroy
- Clone

### 2.2.2 Asynchronous Event APIs

- ServiceEvents

### 2.2.3 Error Handling and Auditing APIs

- GetErrorObject
- DestroyErrorObject
- SetAuditingLevel

### 2.2.4 Security APIs

- CreateSecurityContext
- DestroySecurityContext
- GetSecurityContexts
- Authenticate

## 2.3 GAT Adaptor Registration API

The Adaptor Registration API is language specific and is not covered in this document.

## 2.4 GAT Adaptor Utility API

The Adaptor Utility API is language specific and is not covered in this document.

# 3 GAT Application API Descriptions

This section defines each API call in terms of the information it takes as inputs, the information it outputs, and the status it may flag. No assumption is made about the specifics of how this will be done in any particular language binding – e.g. errors may be simple return codes or may be thrown as exceptions in C++ or Java.

## 3.1 Job APIs

Functions to control job submission and manipulation.

### 3.1.1 SubmitJob

Inputs:

resource resource specification. E.g. this can consist of any of combination of the following

- The name of the operating system
- The type of the operating system
- The version of the operating system
- The release of the operating system
- The name of the host
- The local name of the host
- Number of CPUs.
- The minimum memory required

- The minimum number of CPUs required
- The maximum CPU time in required
- The minimum CPU speed required
- The maximum wall-clock time required

executable The location of the executable to be run.

job type The type of the job – e.g. single, multi, MPI, PVM, ...

arguments List of job arguments

environment List of environment settings

stdin List of (zero or more) files to be used as standard input.

stdout Optional file to be used for standard output.

stderr Optional file to be used for standard error.

Outputs:

job id some handle or id to the new job

Status codes:

GAT\_SUCCESS

This call submits a job to a specified resource or to a resource which matches the criteria.

### 3.1.2 KillJob

Inputs:

job Unique job id of a job.

Outputs:

Status codes:

GAT\_SUCCESS

This call kills a specified job.

### 3.1.3 CheckpointJob

Inputs:

job Id of job to checkpoint.

Outputs:

Status codes:

GAT\_SUCCESS

Tells the specified job to checkpoint.

### 3.1.4 RestoreJob

Inputs:

resource as per SubmitJob  
job Job id.

Outputs:

New job id

Status codes:

GAT\_SUCCESS

Restores the checkpointed job on the new resource.

### 3.1.5 MigrateJob

Inputs:

job Job id of job to migrate  
resource Resource specification as per SubmitJob.

Outputs:

Status codes:

GAT\_SUCCESS

Migrates the specified job to a resource satisfying the new resource criteria

### 3.1.6 GetJobInfo

Inputs:

jobid - a unique identifier for the job to obtain information about.

Outputs:

hostname - The name of the host on which the job is running.  
submission time - The time at which the job was submitted.  
start time - The time at which the job was started.  
stop time - The time at which the job stopped or is expected to stop.

Status codes:

GAT\_SUCCESS

This call obtains information about a specified job.

## 3.2 Resource APIs

### 3.2.1 FindResources

Inputs:

criteria List of resource criteria, as per SubmitJob, plus perhaps network connectivity between them.

Outputs:

resources List of resources satisfying the criteria

Status codes:

GAT\_SUCCESS

Finds resources satisfying the specified criteria.

### 3.2.2 ReserveResources

Inputs:

resource-criteria resource specifications or criteria as in FindResources

time-criteria e.g. minimum time to reservation

Outputs:

reservation-info

Status codes:

GAT\_SUCCESS

Reserves a set of resources given the specified criteria.

### 3.2.3 UnReserveResources

Inputs:

reservation some identifier for the reservation.

Outputs:

Status codes:

GAT\_SUCCESS

Cancels a resource reservation.

### 3.3 Pipe APIs

#### 3.3.1 CreatePipe

Inputs:

Outputs:

pipe

Status codes:

GAT\_SUCCESS

Creates a (bi-directional) pipe. This pipe is initially unconnected and the application must either connect to a remote pipe or set this pipe to be a listener.

#### 3.3.2 DestroyPipe

Inputs:

pipe

Outputs:

Status codes:

GAT\_SUCCESS

Destroys a pipe create by CreatePipe.

#### 3.3.3 ListenOnPipe

Inputs:

pipe

listener A callback-function to be called when a new connection is made to the pipe.

data Data to be passed to the callback function along with a new pipe.

Outputs:

Status codes:

GAT\_SUCCESS

This function makes a pipe listen for incoming connections. On receipt of a new connection the listener function is called with a new pipe and the data argument.

### 3.3.4 ConnectPipe

Inputs:

pipe  
end-point

Outputs:

Status codes:

GAT\_SUCCESS

Connects an existing pipe to the specified end-point.

### 3.3.5 SendToPipe

Inputs:

pipe  
buffer  
buffer-size

Outputs:

number of bytes read This may be less than the buffer size.

Status codes:

GAT\_SUCCESS

Sends the specified number of bytes through the pipe. The call will attempt to deliver all the bytes in the buffer, but on error will return, indicating the number of bytes actually sent.

### 3.3.6 ReceiveFromPipe

Inputs:

pipe  
buffer  
buffer-size

Outputs:

number of bytes received

Status codes:

GAT\_SUCCESS

Reads the specified number of bytes from the pipe, placing them in the buffer. If an error occurs it will return the number of bytes that it actually read.

### 3.4 File APIs

#### 3.4.1 CopyFile

Inputs:

from  
to

Outputs:

Status codes:

GAT\_SUCCESS

Copies a file from one location to another.

#### 3.4.2 MoveFile

Inputs:

from  
to

Outputs:

Status codes:

GAT\_SUCCESS

Moves a file from one location to another.

#### 3.4.3 DeleteFile

Inputs:

location

Outputs:

Status codes:

GAT\_SUCCESS

Deletes the specified file.

### 3.4.4 OpenFile

Inputs:

location File location

Outputs:

file handle

Status codes:

GAT\_SUCCESS

Opens the specified file. This is equivalent to the UNIX open call.

### 3.4.5 CloseFile

Inputs:

file handle

Outputs:

Status codes:

GAT\_SUCCESS

Closes the specified file.

### 3.4.6 ReadFromFile

Inputs:

file handle

buffer

buffer-size

Outputs:

number of bytes read This may be less than the buffer size.

Status codes:

GAT\_SUCCESS

Reads the specified number of bytes from the file.

### 3.4.7 WriteToFile

Inputs:

file handle

buffer

buffer-size

Outputs:

number of bytes written This may be less than the buffer size.

Status codes:

GAT\_SUCCESS

This write the specified number of bytes into the file.

### 3.4.8 SeekInFile

Inputs:

file handle

offset Number of bytes from **whence** to set the new file position.

whence Where to seek from – the start, current position, or end of file.

Outputs:

Status codes:

GAT\_SUCCESS

Sets the current location in the file to the new location specified by the **whence** and **offset** arguments.

### 3.4.9 PositionInFile

Inputs:

file handle

Outputs:

position current position in the file.

Status codes:

GAT\_SUCCESS

Outputs the current position of the file position indicator for the specified file.

### 3.4.10 GetFileInfo

Inputs:

location File to get status of

Outputs:

permissions Does this user have permission to read/write this file.

size How big is the file.

time of last access When was the file last accessed.

Status codes:

GAT\_SUCCESS

The GetFileInfo function returns various pieces of status information about the specified file.

### 3.4.11 AddLocationOfFile

Inputs:

file

location

Outputs:

Status codes:

GAT\_SUCCESS

Associates the specified location with a file.

### 3.4.12 RemoveLocationOfFile

Inputs:

file

location

Outputs:

Status codes:

GAT\_SUCCESS

Removes the specified location of the file.

### 3.4.13 GetLocationsOfFile

Inputs:

file

Outputs:

location list list of locations.

Status codes:

GAT\_SUCCESS

Find all locations of a specified file.

### 3.4.14 ReplicateFileTo

Inputs:

Outputs:

Status codes:

GAT\_SUCCESS

## 3.5 Monitoring APIs

### 3.5.1 GetMetric

Inputs:

metric The metric to be read.

object Any monitorable object.

Outputs:

metric value

Status codes:

GAT\_SUCCESS

GetMetric retrieves a specified measurement associated with any monitorable object, such as a resource, file, pipe, or job.

### 3.5.2 SubscribeToMetric

Inputs:

- metric The metric to be subscribed to.
- object Any monitorable object.
- callback The method to call when the specified metric value is obtained.
- data Arbitrary data object to be passed to the callback function.

Outputs:

Status codes:

GAT\_SUCCESS

Subscribe to the specified metric for the given object. On receipt of new information for this metric, the callback function will be invoked with the metric information and the data object.

### 3.5.3 UnSubscribeFromMetric

Inputs:

- metric The metric to be be unsubscribed from.
- object Any monitorable object.

Outputs:

Status codes:

GAT\_SUCCESS

Unsubscribe from the given metric notification.

## 3.6 Collection APIs

Collections are used to group or to advertise objects. Each collection may contain any number of objects or sub-collections, with associated attributes. One common attribute is an expiry time which is used to remove the object from the collection after it has expired.

### 3.6.1 CreateCollection

Inputs:

- collection ID for new collection

Outputs:

Status codes:

GAT\_SUCCESS

Creates a new collection with the specified ID.

### 3.6.2 DestroyCollection

Inputs:

collection ID of collection to be destroyed

Outputs:

Status codes:

GAT\_SUCCESS

Destroys the given collection.

### 3.6.3 AddToCollection

Inputs:

collection ID of collection

object Object to add to the collection.

attribute list Optional list of attributes for this object.

Outputs:

Status codes:

GAT\_SUCCESS

Adds an object to the collection. The object may be a file, a job, a pipe, resource, information or another collection. An object may have attributes associated with it, one of which may be an expiry time after which the object will be removed from the collection, this expiry time defaults to no-expiry.

### 3.6.4 RemoveFromCollection

Inputs:

collection ID of collection

object Object to remove from the collection.

Outputs:

Status codes:

GAT\_SUCCESS

Removes the given object from the given collection.

### 3.6.5 ListCollection

Inputs:

collection ID of collection

Outputs:

object list List of objects in collection

Status codes:

GAT\_SUCCESS

This lists the objects in the given collection.

### 3.6.6 SetPropertyOfCollection

Inputs:

collection ID of collection

attribute list List of attributes to add to the collection.

Outputs:

Status codes:

GAT\_SUCCESS

Associates a set of attributes with a collection. An object may have many attributes associated with it, one of which may be an expiry time after which the object will be removed from the collection, this expiry time defaults to no-expiry.

### 3.6.7 GetPropertiesOfCollection

Inputs:

collection ID of collection

Outputs:

attribute list List of attributes for collection

Status codes:

GAT\_SUCCESS

Gets all the attributes associated with a collection.

### 3.6.8 FindCollectionByProperties

Inputs:

attribute list List of attributes match collections against.

Outputs:

collection list List of collection IDs

Status codes:

GAT\_SUCCESS

Finds a set of collections based upon its attributes.

### 3.6.9 FindObjectByProperties

Inputs:

collection Collection to search.

attribute list List of attributes match collections against.

Outputs:

object list List of objects

Status codes:

GAT\_SUCCESS

Finds a set of objects in a given collection based upon its attributes.

## 4 GAT Application Utility API

### 4.1 Management APIs

#### 4.1.1 Init

Inputs:

Outputs:

context GAT context

Status codes:

GAT\_SUCCESS

The GAT\_Init call initialises the GAT Engine. The context object returned encapsulates the current state of the Engine and is associated with all GAT operations.

### 4.1.2 Destroy

Inputs:

context GAT context

Outputs:

Status codes:

GAT\_SUCCESS

The Destroy call is used to destroy a GAT Context. If this is the last valid context, the GAT Engine itself will be shut-down.

### 4.1.3 CloneContext

Inputs:

context GAT context

Outputs:

new context

Status codes:

GAT\_SUCCESS

The CloneContext call is used to clone a specified context, copying all state and security information. The new context is completely independent from the original one, which may be destroyed with no effect on the new one.

## 4.2 Asynchronous Event APIs

### 4.2.1 ServiceEvents

Inputs:

context A GAT context

timeout A timeout for the call.

Outputs:

Status codes:

GAT\_SUCCESS

The ServiceEvents call is used to allow the GAT Engine to service asynchronous events. In a single-threaded application it is likely that a timeout would be supplied, in a multi-threaded application one thread may be used for the GAT by using this call and no timeout.

## 4.3 Error Handling and Auditing APIs

### 4.3.1 GetErrorObject

Inputs:

context A GAT context

Outputs:

error object

Status codes:

GAT\_SUCCESS

The GAT\_GetErrorObject call is used to retrieve the error object associated with the last GAT operation.

### 4.3.2 DestroyErrorObject

Inputs:

error object

Outputs:

Status codes:

GAT\_SUCCESS

Once an error object is no longer needed, it should be destroyed.

### 4.3.3 SetAuditingLevel

Inputs:

context A GAT context

level The auditing level.

Outputs:

Status codes:

GAT\_SUCCESS

The GAT Engine may log operations with varying levels of detail. The log of the last operation is accessible via the error object on the GAT Context. The GAT\_SetAuditingLevel call sets the level of logging which is done.

## 4.4 Security APIs

The security APIs are still a work in progress, however they will be similar to the below. The basic idea is that associated with any GAT context is a set of security contexts, which hold the authentication for each VO, or whatever. The application can create a context and authenticate the user, and then any adaptor can retrieve the security contexts from the GAT context and use the appropriate one for contacting external services.

### 4.4.1 CreateSecurityContext

Inputs:

GAT context

Outputs:

security context

Status codes:

GAT\_SUCCESS

Create a security context associated with the given GAT context.

### 4.4.2 DestroySecurityContext

Inputs:

security context

Outputs:

Status codes:

GAT\_SUCCESS

Destroy the specified security context.

### 4.4.3 GetSecurityContexts

Inputs:

GAT context

Outputs:

security context list List of security contexts.

Status codes:

GAT\_SUCCESS

Get a list of security contexts associated with the given GAT context.

#### 4.4.4 Authenticate

Inputs:

security context

credentials Some form of credentials (**TBD**)

Outputs:

Status codes:

GAT\_SUCCESS

Authenticate and attach the authentication with the given security context.

## References

- [1] K.Davis and T.Goodale, ■D1.2 Technical Specification■
- [2] ■Grid Resource Management System■, <http://www.gridlab.org/WorkPackages/wp-9/index.html>
- [3] S.Bradner, ■RFC 2119: Key words for use in RFCs to Indicate Requirement Levels■, <http://www.ietf.org/rfc/rfc2119.txt>
- [4] <http://www.triana.com>