

GT 4.2.1 GRAM4: Developer's Guide

GT 4.2.1 GRAM4: Developer's Guide

Introduction

This guide is intended to help a developer create compatible GRAM4 clients and alternate service implementations.

The key concepts for the GRAM component have not changed. Its purpose is still to provide the mechanisms to execute remote applications for a user. Given an XML job description, GRAM submits the job to a scheduling system such as *PBS* or *Condor*, or to a simple fork-based way of spawning processes, and monitors it until completion. More details can be found here:

<http://www.globus.org/toolkit/docs/3.2/gram/key>

Table of Contents

1. Before you begin	1
1. Feature summary	1
2. Tested platforms	1
3. Backward compatibility summary	2
4. Technology dependencies	2
5. Security Considerations	2
2. Selective Concepts	3
1. Job Submission	3
2. Job status change subscriptions	3
3. Job Termination	4
3. Scenarios	9
1. C	9
2. Java	15
4. Tutorials	22
5. Architecture and design overview	23
1. Job States	23
6. APIs	27
1. Programming Model Overview	27
2. Component API	27
7. Services and WSDL	29
1. Protocol overview	29
2. Operations	29
3. Resource properties	31
4. WSDL and Schema Definition	36
8. Debugging	38
1. Development Logging in Java WS Core	38
2. Enabling debug logging for GRAM classes	38
3. Instrumented timings logging	39
4. Debugging script execution	40
9. Troubleshooting	41
1. Troubleshooting tips	41
2. Java WS Core Errors	42
3. Errors	45
10. Related Documentation	48
11. Internal Components	49
Glossary	50
Index	52

List of Figures

5.1. Managed Executable Job Service Internal State Transition Diagram	26
---	----

List of Tables

9.1. Java WS Core Errors	43
9.2. GRAM4 Errors	46

Chapter 1. Before you begin

1. Feature summary

New Features new since 4.0.x

- New terminate method in the client-side GramJob API
- Improved job lifetime management for users and admins
- Added configuration for "default" Local Resource Managers

Other Standard Supported Features

- Remote job execution and management
- Uniform and flexible interface to batch scheduling systems
- File staging before and after job execution
- File / directory clean up after job execution (after file stage out)
- Service auditing for each submitted

Deprecated Features

- With the addition of the new terminate method in the GramJob API, the destroy method is no longer necessary. For backward compatibility, the destroy method was left in the GramJob API, but it simply calls the terminate method. During the 4.2.x series, clients using the destroy method should change to instead use terminate. In GT 4.4, the plan is to remove the destroy method.

2. Tested platforms

Tested platforms for GRAM4:

- Linux
 - Fedora Core 1 i686
 - Fedora Core 3 i686
 - Fedora Core 3 yup xeon
 - RedHat 7.3 i686
 - RedHat 9 x86
 - Debian Sarge x86
 - Debian 3.1 i686

Tested containers for GRAM4:

- Java WS Core container

3. Backward compatibility summary

Protocol changes since GRAM4 in the GT4.0 series:

- The Java WS Core Framework has been updated from the draft versions of the WSRF/WSN and WS Addressing specifications to the final versions WSRF 1.2, WSN 1.3 and WS Addressing 1.0. There is no backward compatibility between this version and any previous versions.

4. Technology dependencies

GRAM depends on the following GT components:

- Java WS Core
- Transport-Level Security
- Delegation Service
- RFT
- GridFTP
- MDS - internal libraries
- The XML::Parser Perl module is required <http://search.cpan.org/~msergeant/XML-Parser/Parser.pm>

Other scheduler adapters available for GT 4.2.1 release:

- [SGE scheduler adapter interface](#)¹
- IBM LoadLeveler (As of release 3.3.1). For more information see "What's new" in the [LoadLeveler product documentation](#)²
- other batch schedulers... (where the GRAM scheduler interface has been implemented)

5. Security Considerations

No special security considerations exist at this time.

¹ <http://www.lesc.ic.ac.uk/projects/SGE-GT4.html>

² <http://publib.boulder.ibm.com/infocenter/clresctr/index.jsp?topic=/com.ibm.cluster.infocenter.doc/library.html>

Chapter 2. Selective Concepts

This chapter gives an overview and specifics about various topics and concepts of GRAM4.

1. Job Submission

1.1. Client-side generated job ID

A client can generate a job-id and pass it to the call to `ManagedJobFactoryService.createManagedJob()`. This id has been subject of misunderstandings. Please check the section [Client-Side Generated Submission ID](#) in the [User's guide](#) if you have doubts about it.

1.2. Job lifetime

A client can provide a lifetime for a job in job submission. However, sometimes it's hard or impossible to estimate an appropriate lifetime, especially with factors beyond the user's knowledge, like queuing time in the remote local resource manager or load in the remote GT4 container. Please check the following links for information about job lifetime concepts in GRAM4 in GT 4.2:

- [Job Lifetime](#) in the [Execution key concepts](#)
- [Job Lifetime](#) in the [User's guide](#)

2. Job status change subscriptions

2.1. Subscribing for status change notifications

A client can subscribe for job status notifications of a job in two ways

- Subscribe on the call to `ManagedJobFactoryService.createManagedJob()`: GRAM4 creates the subscription resource for the client and returns an EPR of it as part of the response of the `createManagedJob()` call.
- Subscribe in a separate WS call after the call to `ManagedJobFactoryService.createManagedJob()` returned.

The second approach has two disadvantages:

1. Two WS calls as opposed to one WS call in the first approach.
2. There's a risk to miss notifications, because the subscribe call is done after the job had been created.

2.2. Destroying subscription resources

Subscription resources of a job are destroyed automatically on the server-side when the job resource goes away, i.e. when a job resource is destroyed on the server-side. There's no need for the client to manually destroy a subscription resource.

3. Job Termination

3.1. Introduction

Job termination can happen for three reasons:

1. **Processing errors:** Any error that occurs during job processing. This can be an error when files are staged in, an invalid executable, an error of the job in the local resource manager, an error while staging files out, etc.
2. **Job resource expiration:** If the lifetime specified by a client the job will be terminated.
3. **Client cancellation:** A client requests the termination of a job.

If a job is still running and not already fully processed, termination will cause the job to go through a series cleanup steps in GRAM4 before the job-related data is destroyed. The cleanup steps being performed depend on the job and the state it is in. In general this includes cancellation of a running job at the local resource manager and running fileCleanUp if so specified in the job description. Termination at the local resource manager however will only be performed if the job did not already finished executing. This also applies to fileCleanUp: If no fileCleanUp is specified in the job description or if the job already passed fileCleanUp when the termination request comes in, then this step is skipped.

As in normal processing, errors may also happen in the clean up phase. It might e.g. be interesting for a user whether the cancellation of a running job at the local resource manager was successful or not, or whether the specified fileCleanUp had been processed successfully .

This section focuses on the interface GRAM4 offers a client to terminate a job.

In GRAM4 in 4.2 a client cannot request synchronous destruction of a job anymore like it was in GRAM4 in the 4.0 series. The reason for that is that many concurrent destroy requests at a time can cause the GT4 container to become unresponsive. Unlike destroy() the call to terminate() is asynchronous, i.e. returns quickly and does not block until the cleanup steps are all done. The new termination method applies to both ManagedExecutableJobResources (MEJRs) and ManagedMultiJobResources (MMJRs) and is a replacement for the destroy() method in GRAM4 in the 4.0 series.

The new termination method is supposed to cope with different scenarios for jobs in different states, which resulted in an interface that might not be intuitive at first glance. This section explains parameters, return value and faults of the call to terminate() and the implications for jobs in various states.

3.2. User termination method definition

The terminate() method is defined as follows

```
public TerminateOutputType terminate (  
    TerminateInputType parameters)  
    throws RemoteException, ResourceUnknownFaultType,  
    DelegatedCredentialDestroyFaultType,  
    ResourceNotTerminatedFaultType
```

TerminateInputType contains:

- boolean destroyAfterCleanup
- boolean continueNotifying
- boolean destroyDelegatedCredentials

TerminateOutputType contains:

- boolean terminationCompleted

3.3. Explanations for MEJRs

3.3.1. Arguments

destroyAfterCleanup: If set to true the job resource will be destroyed once all cleanup steps are done.

continueNotifying: If set to true a client will be notified about the success of the termination. This happens via a notification bound to the same topic a client subscribes to for normal state change information. A client who is not interested in the success of the termination can set this to false.

destroyDelegatedCredentials: If this is set to true, all delegated credentials that are specified in the job description (if any) will be destroyed after all cleanup steps are done. They must not be destroyed earlier because staging credentials are needed during failureFileCleanUp. Setting this parameter and destroyAfterCleanup both to true enables a client to completely go away after the termination c

3.3.2. Return value

terminationCompleted: Indicates whether termination of the MEJR completed or not when the call to terminate() returns. It is true in case the job is already in a final state, i.e. no cleanup steps had to be done. Otherwise it is false.

If it is true no further notifications will be sent, even if the client requested it, because the job is in a final state where no state transition is happening anymore. If destroyAfterCleanup had been set to true the client can be sure that the job resource has been destroyed when the call to terminate() returns.

If it is false, the job entered the clean up phase and the client can find out about success or failure of the termination either by listening to state change notifications or by querying the status of the MEJR.

3.3.3. Exceptions

ResourceUnknownFaultType: Thrown when the job resource to be terminated does not exist.

DelegatedCredentialDestroyFaultType: Thrown if the client demanded the destruction of delegated credentials and this failed. If this exception is thrown all termination steps succeeded, and only the destruction of the delegated credential failed.

ResourceNotTerminatedFaultType An error occurred during termination. This should not happen at all, but in case unforeseen things happen it will indicate that termination failed.

3.4. Explanations for MMJRs

3.4.1. Arguments

destroyAfterCleanup: If this is set to true the job resource will be destroyed once all cleanup steps are done. This is when the termination calls to all SJs went successfully and the MMJR received final notifications of all SJs.

continueNotifying: If set to true a client will be notified about the success of the termination. This happens via a notification bound to the same topic a client subscribes to for normal state change information. A client who is not interested in the success of the termination can set this to false.

destroyDelegatedCredentials: If this is set to true, all delegated credentials for the MMJR that are specified in the job description will be destroyed when the MMJR is destroyed. They must not be destroyed earlier because job credentials are needed for potential repetitive termination calls. Otherwise the MMJR wouldn't be able to interact with SJ's anymore.

Setting this parameter and `destroyAfterCleanup` both to true enables a client to completely go away after the termination call in case the success of the cleanup steps is not of importance.

3.4.2. Return value

terminationCompleted: Indicates whether termination of the MMJR completed or not. It is true in case the job is already in a final state, i.e. all SJs are in a final state. Otherwise it is false.

If it is true no further notifications will be sent, even if the client requested it, because the job is in a final state where no state transition is happening anymore. If `destroyAfterCleanup` had been set to true the client can be sure that the job resource has been destroyed when the call to `terminate()` returns.

If it is false, the job entered the clean up phase, i.e. termination calls to all subjobs had been sent, and the client can find out about success or failure of the termination either by listening to state change notifications or by querying the status of the MMJR.

3.4.3. Exceptions

ResourceUnknownFaultType: Thrown when the resource to be terminated does not exist.

DelegatedCredentialDestroyFaultType: Thrown if the client demanded the destruction of delegated credentials and this failed. If this exception is thrown all termination steps succeeded, and only the destruction of the delegated credential failed.

ResourceNotTerminatedFaultType: An error occurred during the call to terminate e.g. a termination call to at least one SJ caused a `ResourceNotTerminatedFaultType` exception. Note that a failure in terminating a SJ does not prevent from terminate calls to other SJs. But in this case, the MMJR will not be destroyed after an error from terminating one of the SJs.

3.5. New final job states

Additional to Done and Failed, two new final states `UserTerminateDone` and `UserTerminateFailed` are introduced. These states are part of the notification message sent if a client subscribed for notifications, and they are values of the `ResourceProperty state` that can be queried by a client. Final states and their meaning:

3.5.1. Explanations for MEJRs

Done: The MEJR has been fully processed

Failed: A processing error occurred that resulted in a termination initialized by GRAM4.

UserTerminateDone: The client had called `terminate()` and all cleanup steps have been processed successfully.

UserTerminateFailed: The client had called `terminate()` and at least one cleanup step had not been processed successfully.

3.5.2. Explanations for MMJRs

Done: All SJs are in state Done

Failed: Failed is not necessarily a final state, because a MMJR transitions into state Failed if the first SJ fails. Failed is only a final state if at least one SJ failed and all other SJs are in state Done or Failed. If just one out of N (N>1) SJs failed and the client terminates the MMJR, the state Failed will transition to state UserTerminateDone in case of termination success or to state.

UserTerminateDone: At least one SJ is in state UserTerminateDone and all other SJs are in state UserTerminateDone, Failed or Done.

UserTerminateFailed: At least one SJ is in state UserTerminateFailed and all other SJs are in state UserTerminateDone, UserTerminateFailed, Done or Failed.

3.6. Termination related faults

UserTerminateDone and UserTerminateFailed indicate only whether termination was successful or not. In case of errors in the clean up phase the following faults indicate what had happened. They are part of the notification message to client in case termination failed, and are set in the ResourceProperty `fault`.

These faults will only be set in a resource or are sent to a client as part of a notification message if the clean up steps have to be performed and the job did not already pass the correspondent state. As an example: For a job, that run to completion in the local resource manager, no attempt will be made to cancel it in the local resource manager.

No automated action can be done when these faults are found, they just indicate problems that would have to be manually taken care of.

StagingTerminateFaultType: Interruption of a running transfer failed.

LocalResourceManagerJobTerminateFaultType: Cancellation of the job at the local resource manager failed.

DelegatedCredentialDestroyFaultType: Destroying delegated credentials failed.

3.7. Termination scenarios for MEJRs

Now, all the above might be a bit abstract and hard to understand. Probably the most common scenarios are summarized in this section and illustrate which settings should be chosen in a certain scenario. Since the Java API `GramJob` is commonly used, some of these use-cases refer to it.

1. **A client let `GramJob` delegate and wants to terminate a job and just go away. Information about success of the termination is unimportant.**

```
destroyAfterCleanup=true, continueNotifying=false, destroyDelegatedCredentials=true
```

2. **A client delegated itself and does not want the delegated credential to be destroyed and wants to terminate a job and just go away. Information about success of the termination is unimportant.**

Same like in (1) but `destroyDelegatedCredentials=false`

3. **A client let `GramJob` delegate and wants to terminate a job, get information about the success of termination, subscribed for notifications.**

```
destroyAfterCleanup=false, destroyDelegatedCredentials=true continueNotifying=true
```

If the call to `terminate()` returns true the job has been fully terminated and destroyed. If it returns 'false' the client has to wait for the final notification message of the job (state `UserTerminateDone`, `UserTerminateFailed`). In case

of problems it can find out from the faults that are part of the notification message what happened. In case it does not hear about the job the notification message might have been lost and the client should query the RP 'state' for job status the RP 'fault' to check what happened.

Once termination finished the client should send a second termination call to request destruction of the job resource with parameters `destroyAfterCleanup=true`, `destroyDelegatedCredentials=false` `continueNotifying=false`

4. **A client delegated itself and does not want the delegated credential to be destroyed and wants to terminate a job, get information about the success of termination, subscribed for notifications.**

Same like (3), but `destroyDelegatedCredential=false`

5. **A client let GramJob delegate and wants to terminate a job, get information about the success of termination, didn't subscribe for notifications.**

Same like (3), but periodical querying the RP's state and fault is necessary in case the call to terminate didn't return true.

6. **A client delegated itself and does not want the delegated credential to be destroyed and wants to terminate a job, get information about the success of termination, didn't subscribe for notifications.**

Same like (4), but periodical querying the RP's state and fault is necessary in case the call to terminate didn't return true.

3.8. Repetitive termination

Termination is a kind of one-way street: Once a clean up step is passed there's no way back, and there's no way to repeat a certain step. The only thing GRAM4 provides is information what went wrong by providing the faults along the way. Repetitive termination calls e.g.do not cause repetitive cancellations at the local resource manager.

Repetitive termination calls however can make sense for scenario 3 in the last section: Terminate a job, but don't destroy after clean up to ensure that no information is lost. Then do a second termination call to destroy the job resource.

Chapter 3. Scenarios

1. C

The following is a general scenario for submitting a job using the C stubs and APIs. Please consult the [C WS Core API](#), [GRAM4 API](#) documentation for details on the APIs used in the code excerpts.

1.1. Loading the job description

```
const char *                file = "job.xml";
globus_soap_message_handle_t message;
wsgram_CreateManagedJobInputType1 input;

globus_soap_message_handle_init_from_file(&message, file);

globus_soap_message_deserialize_element_unknown(message, &element);

if(strcmp(element.local, "job") == 0)
{
    wsgram_JobDescriptionType2 *    jd;

    input.choice_value.type = wsgram_CreateManagedJobInputType_job;
    jd = &input.choice_value.value.job;

    wsgram_JobDescriptionType_deserialize(&element, jd, message, 0);
}
else if(strcmp(element.local, "multiJob") == 0)
{
    wsgram_JobDescriptionType3 *    mjd;

    input.choice_value.type = wsgram_CreateManagedJobInputType_multiJob;
    mjd = &input.choice_value.value.multiJob;

    wsgram_MultiJobDescriptionType_deserialize(&element, mjd, message, 0);
}
xsd_QName_destroy_contents(&element);
globus_soap_message_handle_destroy(message);
```

This code sets the choice value of the `wsgram_CreateManagedJobInputType` to be the appropriate type depending on whether the *job description* is a job or *multijob* request.

1.2. Setting the security attributes

```
globus_soap_message_attr_t    message_attr;
```

¹ /api/c/globus_c_gram_client_bindings/html/group_wsgram_CreateManagedJobInputType.html

² /api/c/globus_c_gram_client_bindings/html/group_wsgram_JobDescriptionType.html

³ /api/c/globus_c_gram_client_bindings/html/group_wsgram_MultiJobDescriptionType.html

```

globus_soap_message_attr_init(&message_attr);

/*
 * Set authentication mode to host authorization: other possibilities are
 * GLOBUS_SOAP_MESSAGE_AUTHZ_HOST_IDENTITY or
 * GLOBUS_SOAP_MESSAGE_AUTHZ_HOST_SELF.
 */
globus_soap_message_attr_set(
    message_attr,
    GLOBUS_SOAP_MESSAGE_AUTHZ_METHOD_KEY,
    NULL,
    NULL,
    (void *) GLOBUS_SOAP_MESSAGE_AUTHZ_HOST);

/*
 * Set message protection level. GLOBUS_SOAP_MESSAGE_AUTH_PROTECTION_PRIVACY
 * for encryption.
 */
globus_soap_message_attr_set(
    message_attr,
    GLOBUS_SOAP_MESSAGE_AUTH_PROTECTION_KEY,
    NULL,
    NULL,
    (void *) GLOBUS_SOAP_MESSAGE_AUTH_PROTECTION_PRIVACY);

```

1.3. Creating the factory client handle

```

ManagedJobFactoryService_client_handle_t    factory_handle;

result = ManagedJobFactoryService_client_init(
    &factory_handle,
    message_attr,
    NULL);

```

1.4. Querying for factory resource properties

1.4.1. One at a time

```

/*
 * localResourceManager, or other resource property names as defined in the
 * WSDL
 */
xsd_QName                                property_name =
{
    "http://www.globus.org/namespaces/2008/03/gram/job",
    "localResourceManager"
};
wsrp_GetResourcePropertyResponseType *    property_response;
int                                        fault_type;
xsd_any *                                  fault;

```

```
ManagedJobFactoryPortType_GetResourceProperty(
    factory_handle,
    endpoint,
    &property_name,
    &property_response,
    (ManagedJobFactoryPortType_GetResourceProperty_fault_t *) &fault_type,
    &fault);
```

If this is successful, then `property_response`'s `any` field will contain the deserialized data in the `value` field of the first element in the array.

```
xsd_string * localResourceManager = property_response->any.elements[0].value;

printf("local resource manager is %s\n", *localResourceManager);
```

1.5. Creating the notification consumer

The notification consumer can be either passed in as part of the `wsgam_CreateManagedJobInputType` or through a separate invocation of `ManagedJobPortType_Subscribe_epr()`.

```
globus_service_engine_t          engine;
wsa_EndpointReferenceType        consumer_reference;

globus_service_engine_init(&engine, NULL, NULL, NULL, NULL, NULL);

globus_notification_create_consumer(
    &consumer_reference,
    engine,
    notify_callback,
    NULL);
```

1.6. Creating the job resource

First, prepare the other parts of the `wsgam_CreateManagedJobInputType` structure.

```
/*
 * You can set input.InitialTerminationTime to be a timeout if interested.
 * The xsd_dateTime type is a struct tm pointer.
 */
time_t          term_time = time(NULL);
globus_uuid_t   uuid;
wsa_AttributedURI * job_id;
wsa_EndpointReferenceType * factory_epr;
xsd_any *       reference_property;
wsgam_CreateManagedJobOutputType * output = NULL;
xsd_QName       factory_reference_id_qname =
{
    "http://www.globus.org/namespaces/2008/03/gram/job",
    "ResourceID"
```

```

};

term_time += 60 * 60; /* 1 hour later */
xsd_dateTime_copy(&input.InitialTerminationTime, gmtime(&term_time));

/*
 * Set unique JobID. This is used to reliably create jobs and check for status.
 */
globus_uuid_create(&uuid);
wsa_AttributedURI_init(&job_id);
job_id->base_value = globus_common_create_string("uuid:%s", uuid.text);

/* To subscribe to notifications at create time, add the consumer's EPR to
 * the input message. Otherwise, use the EPR created above in a
 * call to
 */
wsnt_SubscribeType_init(&input.Subscribe);
wsa_EndpointReferenceType_copy_contents(
    &input.Subscribe.ConsumerReference,
    &consumer_reference);

xsd_any_init(&input.Subscribe->TopicExpression.any);
&input.Subscribe->TopicExpression.any->any_info =
    &xsd_QName_contents_info;
xsd_QName_copy(
    (xsd_QName **) &input.Subscribe->TopicExpression.any->any.value,
    &ManagedJobPortType_state_rp_qname);

xsd_anyURI_copy_cstr(
    &input.Subscribe->TopicExpression._Dialect,
    "http://docs.oasis-open.org/wsn/2004/06/TopicExpression/Simple");
xsd_boolean_init(&input.Subscribe->UseNotify);

*(&input.Subscribe->UseNotify) = GLOBUS_TRUE;

/* Construct the EPR of the job factory */
wsa_EndpointReferenceType_init(&factory_epr);
wsa_AttributedURI_init_contents(&factory_epr->Address);
xsd_anyURI_init_contents_cstr(&factory_epr->Address.base_value,
    globus_common_create_string(
        "https://%s:%hu/wsrp/services/%s",
        factory_host,
        factory_port,
        MANAGEDJOBFACTORYSERVICE_BASE_PATH);
wsa_ReferenceParametersTypeinit(&factory_epr->ReferenceParameters);
reference_property = xsd_any_array_push(
    &factory_epr->ReferenceParameters.any);
reference_property->any_info = &xsd_string_info;
xsd_QName_copy(
    &reference_property->element,
    &factory_reference_id_qname);

xsd_string_copy_cstr(
    (xsd_string **) &reference_property->value,

```

```
    "Fork");

/* Submit the request to the service container */
ManagedJobFactoryPortType_createManagedJob_epr(
    factory_handle,
    factory_epr,
    input,
    &output,
    (ManagedJobFactoryPortType_createManagedJob_fault_t *) &fault_type,
    &fault);
```

If this is successful, then the `output` structure will be initialized with the results of the operation. Of particular interest is the `managedJobEndpoint` which contains the reference to the newly-created job resource.

1.7. Subscribing for job state notifications

In order to subscribe for job state change notifications to an existing job resource, initialize the `subscribe_input` used below in the same way as `input.Subscribe` was initialized above.

```
ManagedJobService_client_handle_t    job_handle;
wsnt_SubscribeType                    subscribe_input;
wsnt_SubscribeResponseType *          subscribe_response;

ManagedJobService_client_init(
    &job_handle,
    message_attr,
    NULL);

ManagedJobPortType_Subscribe_epr(
    job_handle,
    output->managedJobEndpoint,
    subscribe_input,
    &subscribe_response,
    (ManagedJobPortType_Subscribe_fault_t *) &fault_type,
    &fault);
```

1.8. Releasing any state holds (if necessary)

```
wsgram_ReleaseInputType                release;
wsgram_ReleaseOutputType *              release_response = NULL;

wsgram_ReleaseInputType_init_contents(&release);

ManagedJobPortType_release_epr(
    job_handle,
    output->managedJobEndpoint,
    &release,
    &release_response,
    (ManagedJobPortType_release_fault_t *) &fault_type,
    &fault);
```

1.9. Destroying resources

```

/* destroy subscription resource */
SubscriptionManagerService_client_init    subscription_handle;
wsnt_DestroyType                          destroy;
wsnt_DestroyResponseType *                destroy_response = NULL;

SubscriptionManagerService_client_init(
    &subscription_handle,
    message_attr,
    NULL);
/* if subscription done at job creation time, use
 * output->subscriptionEndpoint in place of
 * subscribe_response->SubscriptionReference,
 */
SubscriptionManager_Destroy_epr(
    subscription_handle,
    subscribe_response->SubscriptionReference,
    &destroy,
    &destroy_response,
    (SubscriptionManager_Destroy_fault_t *) &fault_type,
    &fault);

/* destroy the job resource */
jobPort.destroy(new Destroy());
ManagedJobPortType_Destroy_epr(
    job_handle,
    output->managedJobEndpoint,
    &destroy,
    &destroy_response,
    (ManagedJobPortType_Destroy_fault_t *) &fault_type,
    &fault);

```

1.10. Building a client

In order to build a client application, certain flags must be passed to the compiler and linker to enable them to be able to locate headers and libraries. The easiest way to do so is to generate a *makefile header*, which is a fragment of a Makefile which includes all of the necessary flags needed to build the application. To do this, issue the command:

```
% globus-makefile-header --flavor=gcc32dbg globus_c_gram_client_bindings > Makefile.inc
```

Then, write your makefile to include this file and use the GLOBUS_CC, GLOBUS_LD, GLOBUS_CFLAGS, GLOBUS_LDFLAGS, and GLOBUS_PKG_LIBS macros. For example:

```

GLOBUS_FLAVOR_NAME=gcc32dbg

include Makefile.inc

CC = $(GLOBUS_CC)
LD = $(GLOBUS_LD)

```

```
CFLAGS = $(GLOBUS_CFLAGS)
LDFLAGS = $(GLOBUS_LDFLAGS) $(GLOBUS_PKG_LIBS)
```

```
client: client.c
```

2. Java

The following is a general scenario for submitting a job using the Java stubs and APIs. Please consult the [Java WS Core API](#), [Delegation API](#), [Reliable File Transfer API](#), and [GRAM4 API](#) documentation for details on classes referenced in the code excerpts.

Also, it will probably be helpful to look at the [GramJob class source code](#)⁴ as a functioning example.

2.1. Class imports

The following imports will be needed for these examples:

```
import java.io.File;5
import java.io.FileInputStream;6
import java.net.URL;7
import java.util.LinkedList;8
import java.util.List;9
import java.util.Vector;10
import java.security.cert.X509Certificate;11
import javax.xml.rpc.Stub;12
import javax.xml.soap.SOAPElement;13
import org.apache.axis.components.uuid.UUIDGenFactory;14
import org.apache.axis.message.addressing.AttributedURI;15
import org.apache.axis.message.addressing.EndpointReferenceType;16
import org.globus.delegation.DelegationUtil;17
import org.globus.exec.generated.CreateManagedJobInputType;18
import org.globus.exec.generated.CreateManagedJobOutputType;19
import org.globus.exec.generated.ManagedJobFactoryPortType;20
import org.globus.exec.generated.ManagedJobPortType;21
```

⁴ http://viewcvs.globus.org/viewcvs.cgi/ws-gram/client/java/source/src/org/globus/exec/client/GramJob.java?rev=1.129.2.3&only_with_tag=globus_4_0_branch&content-type=text/vnd.viewcvs-markup

⁵ <http://java.sun.com/j2se/1.4.2/docs/api/java/io/File.html>

⁶ <http://java.sun.com/j2se/1.4.2/docs/api/java/io/FileInputStream.html>

⁷ <http://java.sun.com/j2se/1.4.2/docs/api/java/net/URL.html>

⁸ <http://java.sun.com/j2se/1.4.2/docs/api/java/util/LinkedList.html>

⁹ <http://java.sun.com/j2se/1.4.2/docs/api/java/util/List.html>

¹⁰ <http://java.sun.com/j2se/1.4.2/docs/api/java/util/Vector.html>

¹¹ <http://java.sun.com/j2se/1.4.2/docs/api/java/security/cert/X509Certificate.html>

¹² <http://java.sun.com/j2ee/1.4/docs/api/javax/xml/rpc/Stub.html>

¹³ <http://java.sun.com/j2ee/1.4/docs/api/javax/xml/soap/SOAPElement.html>

¹⁴ <http://ws.apache.org/axis/java/apiDocs/org/apache/axis/components/uuid/UUIDGenFactory.html>

¹⁵ <http://ws.apache.org/addressing/apidocs/org/apache/axis/message/addressing/AttributedURI.html>

¹⁶ <http://ws.apache.org/addressing/apidocs/org/apache/axis/message/addressing/EndpointReferenceType.html>

¹⁷ http://www.globus.org/api/javadoc-4.0/globus_wsrf_delegation_service_java/org/globus/delegation/DelegationUtil.html

¹⁸ http://www.globus.org/api/javadoc-4.0/globus_wsrf_gram_common_java/org/globus/exec/generated/CreateManagedJobInputType.html

¹⁹ http://www.globus.org/api/javadoc-4.0/globus_wsrf_gram_common_java/org/globus/exec/generated/CreateManagedJobOutputType.html

²⁰ http://www.globus.org/api/javadoc-4.0/globus_wsrf_gram_common_java/org/globus/exec/generated/ManagedJobFactoryPortType.html

²¹ http://www.globus.org/api/javadoc-4.0/globus_wsrf_gram_common_java/org/globus/exec/generated/ManagedJobPortType.html

```

import org.globus.exec.generated.ReleaseInputType;22
import org.globus.exec.utils.ManagedJobConstants;23
import org.globus.exec.utils.ManagedJobFactoryConstants;24
import org.globus.exec.utils.client.ManagedJobClientHelper;25
import org.globus.exec.utils.client.ManagedJobFactoryClientHelper;26
import org.globus.exec.utils.rsl.RSLHelper;27
import org.globus.wsrfl.NotificationConsumerManager;28
import org.globus.wsrfl.WSNConstants;29
import org.globus.wsrfl.encoding.ObjectDeserializer;30
import org.globus.wsrfl.impl.security.authentication.Constants;31
import org.globus.wsrfl.impl.security.authorization.Authorization;32
import org.globus.wsrfl.impl.security.authorization.HostAuthorization;33
import org.globus.wsrfl.impl.security.authorization.IdentityAuthorization;34
import org.globus.wsrfl.impl.security.authorization.SelfAuthorization;35
import org.globus.wsrfl.impl.security.descriptor.ClientSecurityDescriptor;36
import org.globus.wsrfl.impl.security.descriptor.GSISecureMsgAuthMethod;37
import org.globus.wsrfl.impl.security.descriptor.GSITransportAuthMethod;38
import org.globus.wsrfl.impl.security.descriptor.ResourceSecurityDescriptor;39
import org.gridforum.jgss.ExtendedGSSManager;40
import org.oasis.wsn.Subscribe;41
import org.oasis.wsn.SubscribeResponse;42
import org.oasis.wsn.SubscriptionManager;43
import org.oasis.wsn.TopicExpressionType;44
import org.oasis.wsn.WSBaseNotificationServiceAddressingLocator;45
import org.oasis.wsrfl.lifetime.Destroy;46
import org.oasis.wsrfl.properties.GetMultipleResourceProperties_Element;47
import org.oasis.wsrfl.properties.GetMultipleResourcePropertiesResponse;48
import org.oasis.wsrfl.properties.GetResourcePropertyResponse;49

```

²² http://www.globus.org/api/javadoc-4.0/globus_wsrfl_gram_common_java/org/globus/exec/generated/ReleaseInputType.html

²³ http://www.globus.org/api/javadoc-4.0/globus_wsrfl_gram_utils_java/org/globus/exec/utils/ManagedJobConstants.html

²⁴ http://www.globus.org/api/javadoc-4.0/globus_wsrfl_gram_utils_java/org/globus/exec/utils/ManagedJobFactoryConstants.html

²⁵ http://www.globus.org/api/javadoc-4.0/globus_wsrfl_gram_utils_java/org/globus/exec/utils/client/ManagedJobClientHelper.html

²⁶ http://www.globus.org/api/javadoc-4.0/globus_wsrfl_gram_utils_java/org/globus/exec/utils/client/ManagedJobFactoryClientHelper.html

²⁷ http://www.globus.org/api/javadoc-4.0/globus_wsrfl_gram_utils_java/org/globus/exec/utils/rsl/RSLHelper.html

²⁸ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/globus/wsrfl/impl/notification/ClientNotificationConsumerManager.html

²⁹ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/globus/wsrfl/WSNConstants.html

³⁰ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/globus/wsrfl/encoding/ObjectDeserializer.html

³¹ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/globus/wsrfl/impl/security/authentication/Constants.html

³² http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/globus/wsrfl/impl/security/authorization/Authorization.html

³³ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/globus/wsrfl/impl/security/authorization/HostAuthorization.html

³⁴ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/globus/wsrfl/impl/security/authorization/IdentityAuthorization.html

³⁵ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/globus/wsrfl/impl/security/authorization/SelfAuthorization.html

³⁶ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/globus/wsrfl/impl/security/descriptor/ClientSecurityDescriptor.html

³⁷ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/globus/wsrfl/impl/security/descriptor/GSISecureMsgAuthMethod.html

³⁸ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/globus/wsrfl/impl/security/descriptor/GSITransportAuthMethod.html

³⁹ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/globus/wsrfl/impl/security/descriptor/ResourceSecurityDescriptor.html

⁴⁰ http://www.cogkit.org/release/4_1_2/api/jglobus/org/gridforum/jgss/ExtendedGSSManager.html

⁴¹ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/oasis/wsn/Subscribe.html

⁴² http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/oasis/wsn/SubscribeResponse.html

⁴³ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/oasis/wsn/SubscriptionManager.html

⁴⁴ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/oasis/wsn/TopicExpressionType.html

⁴⁵ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/oasis/wsn/WSBaseNotificationServiceAddressingLocator.html

⁴⁶ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/oasis/wsrfl/lifetime/Destroy.html

⁴⁷ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/oasis/wsrfl/properties/GetMultipleResourceProperties_Element.html

⁴⁸ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/oasis/wsrfl/properties/GetMultipleResourcePropertiesResponse.html

⁴⁹ http://www.globus.org/api/javadoc-4.0/globus_java_ws_core/org/oasis/wsrfl/properties/GetResourcePropertyResponse.html

2.2. Loading the job description

```
File jobDescriptionFile = new File("myjobdesc.xml");
JobDescriptionType jobDescription = RSLHelper.readRSL(jobDescriptionFile);
```

The object `jobDescription` will be of sub-type `MultiJobDescriptionType` if the file contents is a multi-job description.

2.3. Creating the factory service stub

```
URL factoryUrl = ManagedJobFactoryClientHelper.getServiceURL(
    contactString).getURL();
String factoryType
    = ManagedJobFactoryConstants.FACTORY_TYPE.<factory type constant>;
EndpointReferenceType factoryEndpoint
    = ManagedJobFactoryClientHelper.getFactoryEndpoint(factoryUrl, factoryType);
ManagedJobFactoryPortType factoryPort
    = ManagedJobFactoryClientHelper.getPort(factoryEndpoint);
```

The format of `contactString` is `[protocol://]host[:port][[/servicepath]]`.

2.4. Loading a proxy from a file

- Default proxy file:

```
ExtendedGSSManager manager =
    (ExtendedGSSManager)ExtendedGSSManager.getInstance();

GSSCredential cred = manager.createCredential(
    GSSCredential.INITIATE_AND_ACCEPT);
```

- Specific proxy file:

```
File proxyFile = new File("proxy_file");
byte[] proxyData = new byte[(int)proxyFile.length];
FileInputStream inputStream = new FileInputStream(proxyFile);
inputStream.read(proxyData);
inputStream.close();

ExtendedGSSManager manager =
    (ExtendedGSSManager)ExtendedGSSManager.getInstance();

GSSCredential proxy = manager.createCredential(
    proxyData,
    ExtendedGSSCredential.IMPEXP_OPAQUE,
    GSSCredential.DEFAULT_LIFETIME,
    null,
    GSSCredential.ACCEPT_ONLY);
```

2.5. Setting stub security parameters

```
ClientSecurityDescriptor secDesc = new ClientSecurityDescriptor();
secDesc.setGSITransport(Constants.<protection level constant>);
secDesc.setAuthz(<Authorization sub-class instance>);
if (proxy != null) {
    secDesc.setGSSCredential(proxy);
}
((Stub) port)._setProperty(Constants.CLIENT_DESCRIPTOR, secDesc);
```

Use setGSISecureMsg() for GSI Secure Message.

2.6. Querying for factory resource properties

2.6.1. One at a time

```
GetResourcePropertyResponse response
    = factoryport.getResourceProperty(ManagedJobConstants.<RP constant>);

SOAPElement[] any = response.get_any();

... = ObjectDeserializer.toObject(any[0], <RP type>.class);
```

2.6.2. Many at a time

```
GetMultipleResourceProperties_Element rpRequest
    = new GetMultipleResourceProperties_Element();
rpRequest.setResourceProperty(new QName[] {
    ManagedJobFactoryConstants.<RP constant #1>,
    ManagedJobFactoryConstants.<RP constant #2>,
    ManagedJobFactoryConstants.<RP constant #N>
});
GetMultipleResourcePropertiesResponse response
    = factoryPort.getMultipleResourceProperties(rpRequest);

SOAPElement[] any = response.get_any();

... = ObjectDeserializer.toObject(any[0], <RP #1 type>.class);
... = ObjectDeserializer.toObject(any[0], <RP #2 type>.class);
... = ObjectDeserializer.toObject(any[0], <RP #N type>.class);
```

2.7. Delegating credentials (if needed)

```
X509Certificate certToSign = DelegationUtil.getCertificateChainRP(
    delegFactoryEndpoint, //EndpointReferenceType
    secDesc, //ClientSecurityDescriptor
)[0]; //first element in the returned array
```

```
EndpointReferenceType credentialEndpoint = DelegationUtil.delegate(
    delegFactoryurl,      //String
    credential,          //GlobusCredential
    certToSign,         //X509Certificate
    lifetime,           //int (seconds)
    fullDelegation,     //boolean
    secDesc);          //ClientSecurityDescriptor
```

There are three types of delegated credentials:

1. Credential used by the job to generate user-owned proxy:

```
jobDescription.setJobCredential(credentialEndpoint);
```

2. Credential used to contact RFT for staging and file clean up:

```
jobDescription.setStagingCredentialEndpoint(credentialEndpoint);
```

3. Credential used by RFT to contact GridFTP servers:

```
TransferRequestType stageOut = jobDescription.getFileStageOut();
stageOut.setTransferCredential(credentialEndpoint);
```

Do the same for fileStageIn and fileCleanUp.

2.8. Creating the job resource

```
CreateManagedJobInputType jobInput = new CreateManagedJobInputType();
jobInput.setJobID(new AttributeURI("uuid: " + UUIDGenFactory.getUUIDGen().nextUUID()));
jobInput.setInitialTerminationTime(<Calendar instance>);
if (multiJob) jobInput.setMultiJob(jobDescription) else jobInput.setJob(jobDescription);
if (subscribeOnCreate) jobInput.setSubscribe(subscriptionReq);
CreateManagedJobOutputType createResponse
    = factoryPort.createManagedJob(jobInput);
EndpointReferenceType jobEndpoint = createResponse.getManagedJobEndpoint();
```

2.9. Creating the job service stub

```
ManagedJobPortType jobPort = ManagedJobClientHelper.getPort(jobEndpoint);
```

You must set the appropriate security parameters for the job service stub (jobPort) as well.

2.10. Subscribing for job state notifications

```
NotificationConsumerManager notifConsumerManager
    = NotificationConsumerManager.getInstance();
```

```
notifConsumerManager.startListening();
List topicPath = new LinkedList();
```

```

topicPath.add(ManagedJobConstants.RP_STATE);

ResourceSecurityDescriptor resourceSecDesc = new ResourceSecurityDescriptor();
resourceSecDesc.setAuthz(Authorization.<authz type constant>);

Vector authMethods = new Vector();
authMethods.add(GSITransportAuthMethod.BOTH);
resourceSecDesc.setAuthMethods(authMethods);

EndpointReferenceType notificationConsumerEndpoint
    = notifConsumerManager.createNotificationConsumer(
        topicPath,
        this,
        resourceSecDesc);

Subscribe subscriptionReq = new Subscribe();
subscriptionReq.setConsumerReference(
    notificationConsumerEndpoint);

TopicExpressionType topicExpression = new TopicExpressionType(
    WSNConstants.SIMPLE_TOPIC_DIALECT,
    ManagedJobConstants.RP_STATE);
subscriptionReq.setTopicExpression(topicExpression);

EndpointReferenceType subscriptionEndpoint;

```

- Subscribe on creation

```
jobInput.setSubscribe(subscriptionReq);
```

- Subscribe after creation

```

SubscribeResponse subscribeResponse
    = jobPort.subscribe(subscriptionRequest);
subscriptionEndpoint = subscribeResponse.getSubscriptionReference();

```

2.11. Releasing any state holds (if necessary)

```
jobPort.release(new ReleaseInputType());
```

2.12. Destroying resources

```

/*destroy subscription resource*/
SubscriptionManager subscriptionManagerPort
    = new WSBaseNotificationServiceAddressingLocator()
        .getSubscriptionManagerPort(subscriptionEndpoint);

//set stub security parameters on subscriptionManagerPort

```

```
subscriptionManagerPort.destroy(new Destroy());
```

```
/*destroy the job resource*/  
jobPort.destroy(new Destroy());
```

Chapter 4. Tutorials

The following tutorials are available for GRAM4 developers:

- [GRAM4 Scheduler Interface Tutorial](#)¹

¹ scheduler-tutorial.html

Chapter 5. Architecture and design overview

The GRAM services in GT 4.2.1 are WSRF compliant. One of the key concepts in the WSRF¹ specification is the decoupling of a service with the public "state" of the service in the interface via the implied resource pattern². Following this concept, the data of GT 4.2.1 GRAM jobs is published as part of WSRF resources, while there is only one service to start jobs or query and monitor their state. This is different from the OGSI³ model of GT3 where each job was represented as a separate service. There is still a job factory service that can be called in order to create job instances (represented as WSRF resources). Each scheduling system that GRAM is interfaced with is represented as a separate factory resource. By making a call to the factory service while associating the call to the appropriate factory resource, the job submitting actor can create a job resource mapping to a job in the chosen scheduling system.

1. Job States

1.1. Overview

The *Managed Executable Job Service (MEJS)* relies on a state machine to handle state transitions. There are two sets of states: external and internal. The external states are those that the user gets in notifications and can be queried as a resource property. The internal states are those that are strictly used by the state machine to step through all the necessary internal tasks that need to be performed for a particular job.

The Managed Multi Job Service does not rely on a state machine, but instead makes judgements after receiving notifications from the sub-jobs about which external state it should be in. The external states for the *MMJS* are identical to the ones used by the MEJS.

1.2. External and Internal States of the Managed Job Services

1.2.1. External States of the Managed Job Services

- Unsubmitted
- StageIn
- Pending
- Active
- Suspended
- StageOut
- CleanUp
- Done

¹ <http://www.globus.org/wsrf/>

² <http://www.globus.org/wsrf/faq.php#wsrf12>

³ http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-04_2002-10-04.pdf

- Failed
- UserTerminateDone
- UserTerminateFailed

1.2.2. Internal States of the Managed Executable Job Service

- None
- Start
- StageIn
- StageInHold
- StageInResponse
- Submit
- PendingHold
- WaitingForStateChanges
- Suspend
- Resume
- OpenStdout
- OpenStderr
- MergeStdout
- StageOut
- StageOutHold
- StageOutResponse
- CleanUp
- CleanUpHold
- FileCleanUp
- FileCleanUpResponse
- CacheCleanUp
- UserTerminate
- SystemTerminate
- FailureFileCleanUp
- FailureFileCleanUpResponse

- FailureCacheCleanUp
- FinalizeTermination
- Done
- Restart

1.3. Managed Executable Job Service Internal State

Here is a diagram illustrating the internal state transitions of the Managed Executable Job Service and how the external states are triggered within this progression:

Figure 5.1. Managed Executable Job Service Internal State Transition Diagram

Chapter 6. APIs

1. Programming Model Overview

This component consists abstractly of two interfaces: the Managed Job Factory Port Type (MJFPT) and the Managed Job Port Type (MJPT).

In actuality there are three service/resource implementations, two of which implement the basic MJPT. The first one is the service which actually talks to a particular local resource manager to execute a process on the remote computer or cluster. This one is called a *Managed Executable Job Service (MEJS)* and its resource is called the Managed Executable Job Resource (MEJR). The second is a special implementation which accepts a multi-job description, breaks the description up into single-job descriptions, and then submits each of these so-called "sub-jobs" to an MEJS. This implementation is called the *Managed Multi Job Service (MMJS)*. Its resource is called the Managed Multi-Job Resource (MMJR)

Because of the fact that these two job services use the same port type, the API for accessing both the MEJR and the MMJR are identical. The *MJFS* creates the appropriate job resource depending on the factory resource used to qualify the operation call. Most of the factory resources represent local resource managers used by the MEJS (*PBS*, *LSF*, *Condor*). There is a special Multi factory resource which represents an abstract multi-job resource manager. The appropriate *job description* type is required for the two different types of managed job.

2. Component API

Java API Documentation Links (Javadoc)

- [Client API](#)¹
- [Auto-Generated Service Stubs and Persistence Data Objects API](#)²
- [Service API](#)³
- [Utilities API](#)⁴
- [Job Monitoring API](#)⁵

C API Documentation Links

- [All C APIs](#)⁶
- [GRAM4 Client Bindings](#)⁷ [[noframes](#)⁸]
- [WS-Rendezvous Client Bindings](#)⁹ [[noframes](#)¹⁰]

¹ http://www.globus.org/api/javadoc-4.0.0/globus_wsrf_gram_client_java

² http://www.globus.org/api/javadoc-4.0.0/globus_wsrf_gram_common_java

³ http://www.globus.org/api/javadoc-4.0.0/globus_wsrf_gram_service_java

⁴ http://www.globus.org/api/javadoc-4.0.0/globus_wsrf_gram_utils_java

⁵ http://www.globus.org/api/javadoc-4.0.0/globus_wsrf_gram_job_monitoring_common_java

⁶ <http://www.globus.org/api/c-globus-4.0/>

⁷ http://www.globus.org/api/c-globus-4.0/globus_c_gram_client_bindings/html/index.html

⁸ http://www.globus.org/api/c-globus-4.0/globus_c_gram_client_bindings/html/modules.html

⁹ http://www.globus.org/api/c-globus-4.0/globus_c_rendezvous_client_bindings/html/index.html

¹⁰ http://www.globus.org/api/c-globus-4.0/globus_c_rendezvous_client_bindings/html/modules.html

- [GRAM4 Scheduler Event Generator](#)¹¹ [[noframes](#)¹²]

¹¹ http://www.globus.org/api/c-globus-4.0/globus_scheduler_event_generator/html/index.html

¹² http://www.globus.org/api/c-globus-4.0/globus_scheduler_event_generator/html/modules.html

Chapter 7. Services and WSDL

1. Protocol overview

GRAM4 allows for remote execution and management of programs through the creation of a managed job. The management of the job is taken care of primarily by core toolkit functionality (WS-ResourceLifetime and WS-BaseN implementations). Please see [Java WS Core](#) on notifications and resource lifetime (destruction) for more information.

1.1. Managed Job Factory Service (MJFS)

A single MJFS is used to create all jobs for all users. For each local resource manager, a dedicated Managed Job Factory Resource (MJFR) enables the MJFS to publish information about the characteristics of the compute resource, for example:

- host information
- GridFTP URL (for file staging and streaming)
- compute cluster size and configuration, and so on...

In addition, there is a special MJFR which is used for creating MMJRs.

1.2. Managed Executable Job Service (MEJS)

A single *MEJS* is used to manage all executable jobs for all users. Each Managed Executable Job Resource (MEJR) enables the MEJS to publish information about the individual job the MEJR represents. This information can be accessed by querying the MEJS for the resource properties of a given MEJR, such as the:

- current job state
- stdout location
- stderr location
- exit code, and so on.

1.3. Managed Multi Job Service (MMJS)

A single MMJS is used to manage all multi-jobs for all users. Each Managed Multi-Job Resource (MMJR) enables the MMJS to publish information about the individual multi-job the MMJR represents. This information can be accessed by querying the MMJS for the resource properties of a given MMJR, such as the:

- current overall job state
- list of sub-job EPRs

2. Operations

There are just two operations defined in the GRAM port types (not counting the Rendezvous port type which is used for MPI job synchronization): "createManagedJob" in the Managed Job Factory port type, and "release" in the Managed

Job port type. All other operations (such as canceling/killing the job and querying for resource properties) are provided by the underlying WSRF implementation of the toolkit.

2.1. ManagedJobFactoryPortType

- `createManagedJob`: This operation creates either a MEJR or MMJR, subscribes the client for notifications if requested, and replies with one or two endpoint references (EPRs). The input of this operation consists of a *job description*, an optional initial termination time for the job resource, and an optional state notification subscription request.

The first EPR:

- is qualified with the identifier to the newly created MEJR or MMJR
- points to either the MEJS or MMJS.

The second EPR:

- is only present if a notification subscription was requested
- is qualified with the identifier to the newly created subscription resource
- points to the subscription manager service.

Using the optional subscription request provides an efficient means of subscribing to the newly created MEJR or MMJR without additional round-trip messages. Clients who subscribe afterwards must check the current status of the job, since the inherent race-condition means some state-changes may have occurred prior to the separate subscription request. In any event, there is a slight risk of lost notifications due to the lack of reliability guarantees in the notification delivery mechanism from WS-BaseNotification.

The `ManagedJobFactoryPortType` also has all the operations and publishes all the resource properties (via the MJFR) defined in the following `WS-ResourceProperties`¹ port types:

- `GetResourceProperty`
- `GetMultipleResourceProperties`
- `QueryResourceProperties`

2.2. ManagedJobPortType

This port type does not define any new operations itself, but has all the operations and publishes all the resource properties defined in the following port types:

ReleaseManagedJob port type:

- `release`: This operation takes no parameters and returns nothing. Its purpose is to release a hold placed on a state through the use of the "holdState" field in the job description. See the [do-main-specific GRAM4 component documentation](#)² for more information on the "holdState" field.

TerminateManagedJob port type:

¹ <http://www.ibm.com/developerworks/library/ws-resource/ws-resourceproperties.pdf>

² [../schemas/gram_job_description.html](#)

- `terminate`: This operation terminates a job. Depending on arguments and the state of the job this may result in immediate destruction of the job resource or in starting of clean up steps and resource destruction after the clean up is done.

*WS-ResourceProperties*³ port types:

- `GetResourceProperty`
- `GetMultipleResourceProperties`
- `QueryResourceProperties`

*WS-ResourceLifetime*⁴ port types:

- `ScheduledResourceTermination`

*WS-BaseNotification*⁵ port type:

- `NotificationProducer`

2.3. Managed Executable Job Port Type

This port type does not define any new operations. See "Resources Properties" under [Services and WSDL](#).

2.4. Managed Multi-Job Port Type

This port type does not define any new operations. See "Resources properties" below.

3. Resource properties

3.1. Managed Job Factory Port Type

- `{http://www.globus.org/namespaces/2008/03/gram/job}condorArchitecture`
Condor architecture label.
- `{http://www.globus.org/namespaces/2008/03/gram/job}condorOS`
Condor OS label.
- `{http://www.globus.org/namespaces/2008/03/gram/job}delegationFactoryEndpoint`
The endpoint reference to the delegation factory used to delegated credentials to the job.
- `{http://mds.globus.org/glue/ce/1.1}GLUECE`
GLUE data
- `{http://mds.globus.org/glue/ce/1.1}GLUECESummary`
GLUE data summary

³ <http://www.ibm.com/developerworks/library/ws-resource/ws-resourceproperties.pdf>

⁴ <http://www.ibm.com/developerworks/library/ws-resource/ws-resourcelifetime.pdf>

⁵ <ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-BaseN.pdf>

- `{http://www.globus.org/namespaces/2008/03/gram/job}globusLocation`
The location of the Globus Toolkit installation that these services are running under.
- `{http://www.globus.org/namespaces/2008/03/gram/job}hostCPUType`
The job host CPU architecture (i686, x86_64, etc...)
- `{http://www.globus.org/namespaces/2008/03/gram/job}hostManufacturer`
The host manufacturer name. May be "unknown".
- `{http://www.globus.org/namespaces/2008/03/gram/job}hostOSName`
The host OS name (Linux, Solaris, etc...)
- `{http://www.globus.org/namespaces/2008/03/gram/job}hostOSVersion`
The host OS version.
- `{http://www.globus.org/namespaces/2008/03/gram/job}localResourceManager`
The local resource manager type (i.e. *Condor*, Fork, *LSF*, Multi, *PBS*, etc...)
- `{http://www.globus.org/namespaces/2008/03/gram/job}availableLocalResourceManager`
All local resource managers that are configured in this GRAM4 instance
- `{http://www.globus.org/namespaces/2008/03/gram/job}jobTTLAfterProcessing`
Time in seconds a job resource will stay alive after a job finished processing in GRAM4 (including fileStageOut, fileCleanUp). When this time elapsed the job resource is destroyed and no longer be available for a client. A negative values means that the job resource will never be destroyed.
- `{http://www.globus.org/namespaces/2008/03/gram/job}maxJobLifetime`
Max time in seconds a user can set as initial lifetime in job submission or in subsequent setTerminationTime calls. A negative value means that there is no limit.
- `{http://mds.globus.org/metadata/2005/02}ServiceMetaDataInfo`
service start time, Globus Toolkit(R) version, service type name
- `{http://www.globus.org/namespaces/2008/03/gram/job}scratchBaseDirectory`
The directory recommended by the system administrator to be used for temporary job data.
- `{http://www.globus.org/namespaces/2008/03/gram/job}stagingDelegationFactoryEndpoint`
The endpoint reference to the delegation factory used to delegated credentials to the staging service (RFT).

3.2. Managed Job Port Type

- `{http://www.globus.org/namespaces/2008/04/rendezvous}Capacity`

Used for Rendezvous.

- `{http://docs.oasis-open.org/wsrf/r1-2}CurrentTime`
Time of creation.
- `{http://docs.oasis-open.org/wsrf/rp-2}QueryExpressionDialect`
From the QueryResourceProperties port type.
- `{http://www.globus.org/namespaces/2008/03/gram/job/faults}fault`
Faults (if generated) that happen along job processing and that cause a job to fail.
- `{http://www.globus.org/namespaces/2008/03/gram/job/types}holding`
Indicates whether a hold has been placed on this job.
- `{http://www.globus.org/namespaces/2008/03/gram/job/types}localUserId`
The job owner's local user account name.
- `{http://www.globus.org/namespaces/2008/04/rendezvous}RegistrantData`
Used for Rendezvous.
- `{http://www.globus.org/namespaces/2008/04/rendezvous}RendezvousCompleted`
Used for Rendezvous.
- `{http://www.globus.org/namespaces/2008/03/gram/job/description}service-LevelAgreement`
A wrapper around fields containing the single-job and multi-job descriptions or *RSLs*. Only one of these sub-fields shall have a non-null value.
- `{http://www.globus.org/namespaces/2008/03/gram/job/types}state`
The current state of the job.
- `{http://docs.oasis-open.org/wsrf/r1-2}TerminationTime`
Time when the resource expires.
- `{http://www.globus.org/namespaces/2008/03/gram/job/types}userSubject`
The GSI certificate DN of the job owner.

3.3. Managed Executable Job Port Type

- `{http://docs.oasis-open.org/wsrf/r1-2}CurrentTime`
Time of creation.
- `{http://docs.oasis-open.org/wsrf/r1-2}TerminationTime`
Time when the resource expires.

- `{http://www.globus.org/namespaces/2008/03/gram/job/exec}credentialPath`
The path (relative to the job process) to the file containing the user proxy used by the job to authenticate out to other services.
- `{http://www.globus.org/namespaces/2008/03/gram/job/types}exitCode`
The exit code generated by the job process.
- `{http://www.globus.org/namespaces/2008/03/gram/job/faults}fault`
The fault (if generated) indicating the reason for failure of the job to complete.
- `{http://www.globus.org/namespaces/2008/03/gram/job/types}holding`
Indicates whether a hold has been placed on this job.
- `{http://www.globus.org/namespaces/2008/03/gram/job/types}localUserId`
The job owner's local user account name.
- `{http://www.globus.org/namespaces/2008/03/gram/job/exec}localJobId`
The job id(s) of the job in the local resource manager. Note that for Fork jobs these id's are prefixed with the uuid of the job.
- `{http://www.globus.org/namespaces/2008/03/gram/job/description}service-LevelAgreement`
A wrapper around fields containing the single-job and multi-job descriptions or *RSLs*. Only one of these sub-fields shall have a non-null value.
- `{http://www.globus.org/namespaces/2008/03/gram/job/types}state`
The current state of the job.
- `{http://www.globus.org/namespaces/2008/03/gram/job/exec}stderrURL`
A GridFTP URL to the file generated by the job which contains the stderr.
- `{http://www.globus.org/namespaces/2008/03/gram/job/exec}stdoutURL`
A GridFTP URL to the file generated by the job which contains the stdout.
- `{http://www.globus.org/namespaces/2008/03/gram/job/types}userSubject`
The GSI certificate DN of the job owner.
- `{http://www.globus.org/namespaces/2008/04/rendezvous}Capacity`
Used for Rendezvous.
- `{http://www.globus.org/namespaces/2008/04/rendezvous}RegistrantData`
Used for Rendezvous.
- `{http://www.globus.org/namespaces/2008/04/rendezvous}RendezvousCompleted`

Used for Rendezvous.

- `{http://docs.oasis-open.org/wsrf/rp-2}QueryExpressionDialect`

From the QueryResourceProperties port type.

3.4. Managed Multi-Job Port Type

- `{http://docs.oasis-open.org/wsrf/r1-2}CurrentTime`

Time of creation.

- `{http://docs.oasis-open.org/wsrf/r1-2}TerminationTime`

Time when the resource expires.

- `{http://www.globus.org/namespaces/2008/03/gram/job/faults}fault`

The fault (if generated) indicating the reason for failure of the job to complete.

- `{http://www.globus.org/namespaces/2008/03/gram/job/types}holding`

Indicates whether a hold has been placed on this job.

- `{http://www.globus.org/namespaces/2008/03/gram/job/types}localUserId`

The job owner's local user account name.

- `{http://www.globus.org/namespaces/2008/03/gram/job/description}service-LevelAgreement`

A wrapper around fields containing the single-job and multi-job descriptions or *RSLs*. Only one of these sub-fields shall have a non-null value.

- `{http://www.globus.org/namespaces/2008/03/gram/job/types}state`

The current state of the job.

- `{http://www.globus.org/namespaces/2008/03/gram/job/multi}subJobEndpoint`

A set of endpoint references to the sub-jobs created by this multi-job.

- `{http://www.globus.org/namespaces/2008/03/gram/job/types}userSubject`

The GSI certificate DN of the job owner.

- `{http://www.globus.org/namespaces/2008/04/rendezvous}Capacity`

Used for Rendezvous.

- `{http://www.globus.org/namespaces/2008/04/rendezvous}RegistrantData`

Used for Rendezvous.

- `{http://www.globus.org/namespaces/2008/04/rendezvous}RendezvousCompleted`

Used for Rendezvous.

- `{http://docs.oasis-open.org/wsrf/rp-2}QueryExpressionDialect`

From the QueryResourceProperties port type.

4. WSDL and Schema Definition

WSDL links:

- [ManagedJobFactoryPortType](#)⁶
- [ManagedJobPortType](#)⁷
- [ReleaseManagedJobPortType](#)⁸
- [TerminateManagedJobPortType](#)⁹
- [ManagedExecutableJobPortType](#)¹⁰
- [ManagedMultiJobPortType](#)¹¹

Schema links:

- [CAS Types](#)¹²
- [File System Mapping Config Schema](#)¹³
- GLUE Schema
 - [Batch Providers](#)¹⁴
 - [Compute Element](#)¹⁵
 - [Metadata](#)¹⁶
- [Job Description Schema](#)¹⁷
- [Managed Job Faults Schema](#)¹⁸
- [Managed Job Types Schema](#)¹⁹
- [RFT Types Schema](#)²⁰

⁶ http://viewcvs.globus.org/viewcvs.cgi/ws-gram/common/schema/gram/jdd/managed_job_factory_port_type.wsdl?revision=1.6

⁷ http://viewcvs.globus.org/viewcvs.cgi/ws-gram/common/schema/gram/jdd/managed_job_port_type.wsdl?revision=1.9

⁸ http://viewcvs.globus.org/viewcvs.cgi/ws-gram/common/schema/gram/jdd/release_managed_job_provider_port_type.wsdl?revision=1.2

⁹ http://viewcvs.globus.org/viewcvs.cgi/ws-gram/common/schema/gram/jdd/terminate_managed_job_provider_port_type.wsdl?revision=1.3

¹⁰ http://viewcvs.globus.org/viewcvs.cgi/ws-gram/common/schema/gram/jdd/managed_executable_job_port_type.wsdl?revision=1.9

¹¹ http://viewcvs.globus.org/viewcvs.cgi/ws-gram/common/schema/gram/jdd/managed_multi_job_port_type.wsdl?revision=1.9

¹² [../schemas/cas_types.html](#)

¹³ [../schemas/gram_fs_map.html](#)

¹⁴ [../schemas/batchproviders.html](#)

¹⁵ [../schemas/glue_ce.html](#)

¹⁶ [../schemas/metadata.html](#)

¹⁷ [../schemas/gram_job_description.html](#)

¹⁸ [../schemas/mj_faults.html](#)

¹⁹ [../schemas/mj_types.html](#)

²⁰ [../schemas/rft_types.html](#)

- [WS Addressing Schema](#)²¹
- [WS Base Faults Schema](#)²²

²¹ ../schemas/ws_addressing.html
²² ../schemas/ws_base_faults.html

Chapter 8. Debugging

Log output from GRAM4 is a useful tool for debugging issues. Because GRAM4 is built on top of Java WS Core, developer debugging is the same as described in [Chapter 10, Debugging](#). For sys admin logging information, see [Chapter 10, Admin Debugging](#).

1. Development Logging in Java WS Core

The following information applies to Java WS Core and those services built on it.

Logging in the Java WS Core is based on the [Jakarta Commons Logging](#)¹ API. Commons Logging provides a consistent interface for instrumenting source code while at the same time allowing the user to plug-in a different logging implementation. Currently we use [Log4j](#)² as a logging implementation. Log4j uses a separate configuration file to configure itself. Please see Log4j documentation for details on the [configuration file format](#)³.

1.1. Configuring server side developer logs

Server side logging can be configured in `$GLOBUS_LOCATION/container-log4j.properties`, when the container is stand alone container. For tomcat level logging, refer to [Logging for Tomcat](#)⁴, . The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

Additional logging can be enabled for a package by adding a new line to the configuration file. Example:

```
#for debug level logging from org.globus.package.FooClass
log4j.category.org.globus.package.name.FooClass=DEBUG
#for warnings from org.some.warn.package
log4j.category.org.some.warn.package=WARN
```

1.2. Configuring client side developer logs

Client side logging can be configured in `$GLOBUS_LOCATION/log4j.properties`. The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

2. Enabling debug logging for GRAM classes

To turn on debug logging for the [Managed Executable Job Service \(MEJS\)](#), add the following entry to the `container-log4j.properties` file:

```
log4j.category.org.globus.exec.service.exec=DEBUG
```

¹ <http://jakarta.apache.org/commons/logging/>

² <http://logging.apache.org/log4j/>

³ [http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure\(java.lang.String,org.apache.log4j.spi.LoggerRepository\)](http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure(java.lang.String,org.apache.log4j.spi.LoggerRepository))

⁴ <http://tomcat.apache.org/tomcat-5.5-doc/logging.html>

To turn on debug logging for the delegated proxy management code, add the following entry to the `container-log4j.properties` file:

```
log4j.category.org.globus.exec.service.utils=DEBUG
```

To turn on debug logging for the *Managed Multi Job Service (MMJS)*, add the following entry to the `container-log4j.properties` file:

```
log4j.category.org.globus.exec.service.multi=DEBUG
```

To turn on debug logging for the *Managed Job Factory Service (MJFS)*, add the following entry to the `container-log4j.properties` file:

```
log4j.category.org.globus.exec.service.factory=DEBUG
```

To turn on debug logging for all GRAM code, add the following entry to the `container-log4j.properties` file:

```
log4j.category.org.globus.exec=DEBUG
```

Follow the pattern to turn on logging for other specific packages or classes.

3. Instrumented timings logging

Both the service and Java client API code contain special debugging statements which output certain timing data to help in determining performance bottlenecks.

The service code uses the `PerformanceLog` class to output the timings information. To turn on service timings logging without triggering full debug logging for the service code, add the following lines to the `container-log4j.properties` file:

```
log4j.category.org.globus.exec.service.factory.ManagedJobFactoryService.performance=DEBUG
log4j.category.org.globus.exec.service.exec.ManagedExecutableJobResource.performance=DEBUG
log4j.category.org.globus.exec.service.exec.StateMachine.performance=DEBUG
```

The Java client API has not been converted over to using the `PerformanceLog` class, so the debug statements are sent at the `INFO` level to avoid having to turn on full debug logging. To turn on client timings logging without triggering full debug logging for the client code, add the following line to the `container-log4j.properties` file:

```
log4j.category.org.globus.exec.client.e=INFO
```

There are two parsing scripts available in the source distribution that aren't distributed in any GPT package for summarizing the service and client timings data. They are located in `ws-gram/service/java/test/throughput/`, and are named `parse-service-timings.pl` and `parse-client-timings.pl`. They both simply take the path of the appropriate log file that contains the timing data. These scripts work fine with log files that have other logging statements mixed with the timing data.

4. Debugging script execution

It may be necessary to debug the *scheduler* scripts if jobs aren't being submitted correctly, and either no fault or a less-than-helpful fault is generated. Ideally we would like that this not be necessary; so if you find that you must resort to this, please file a bug report or let us know on the discuss e-mail list.

By turning on debug logging for the MEJS (see above), you should be able to search for "*Perl Job Description*" in the logging output to find the perl form of the *job description* that is sent to the scheduler scripts.

Also by turning on debug logging for the MEJS, you should be able to search for "*Executing command*" in the logging output to find the specific commands that are executed when the scheduler scripts are invoked from the service code. If you saved the perl job description from the previous paragraph, then you can use this to manually run these commands.

There is a perl job description attribute named `logfile` that isn't currently supported in the XML job description that can be used to print debugging info about the execution of the perl scripts. The value for this attribute is a path to a file that will be created. You can add this to the perl job description file that you created from the service debug logging before manually running the script commands.

Beyond the above advice, you may want to edit the perl scripts themselves to print more detailed information. For more information on the location and composition of the scheduler scripts, please consult the [GRAM4 Scheduler Interface Tutorial](#)⁵.

⁵ ../developer/scheduler-tutorial-perl.html

Chapter 9. Troubleshooting

For a list of common errors in GT, see [Error Codes](#).

For information about sys admin logging, see [Chapter 10, Admin Debugging](#) in the GRAM4 Admin Guide.

1. Troubleshooting tips

In case you run into problems you can do the following

- Check the GRAM4 documentation. Maybe you'll find hints here to solve your problem.
- Send e-mails to one of several Globus e-mail lists. You'll have to subscribe to a list before you can send an e-mail to it. See [here](#)¹ for general e-mail lists and information on how to subscribe to a list and [here](#)² for GRAM specific lists.

Probably the best lists for GRAM4-related problems are `gt-user@globus.org` and `gram-user@globus.org`

- Check the container log for errors.

In case you don't find anything suspicious you can increase the log-level of GRAM4 or other relevant components. Maybe the additional logging-information will tell you what's going wrong. General information about container logging can be found [Logging in Java WS Core](#) section.

To get debug information from GRAM4, un-comment the following line in `$GLOBUS_LOCATION/container-log4j.properties` by removing the leading '#' and restart the GT4 server.

```
# log4j.category.org.globus.exec=DEBUG
```

The logging output can either be found on the console if you started the container using `globus-start-container` (maybe with arguments) or in `$GLOBUS_LOCATION/var/container.log` in if you started the container using the command `globus-start-container-detached`

¹ http://dev.globus.org/wiki/Mailing_Lists

² http://dev.globus.org/wiki/GRAM#Mailing_Lists

2. Java WS Core Errors

Table 9.1. Java WS Core Errors

Error Code	Definition
Failed to acquire notification consumer home instance from registry	Caused by <code>javax.naming.NameNotFoundException</code> : Name <code>services</code> is not bound in
The WS-Addressing 'To' request header is missing	This warning is logged by the container if the request did not contain the necessary <i>WS-Addressing</i> headers, those headers at all or is somehow misconfigured.
java.io.IOException: Token length X > 33554432	If you see this error in the container log, it usually means you are trying to connect to HTTPS server using <code>https</code> specifies 8443 as a port number and <code>http</code> as the protocol name.
java.lang.NoSuchFieldError: DOCUMENT	This error usually indicates a mismatch between the version of Apache Axis that the code was compiled with and the version currently running with.
org.globus.wsrfl.InvalidResourceKeyException: Argument key is null / Resource key is missing	These errors usually indicate that a resource key was not passed with the request or that an invalid resource key was used (the element <code>QName</code> of the resource key did not match what the service expected).
Unable to connect to localhost:xxx	Cannot resolve localhost. The machine's <code>/etc/hosts</code> isn't set up correctly and/or you do not have DNS for
org.globus.common.ChainedIOException: Failed to initialize security context	This may indicate that the user's proxy is invalid.
Error: org.xml.sax.SAXException: Unregistered type: class xxx	This may indicate that an Axis generated XML type, defined by the WS RLS XSD, was not properly registered upon deployment without intervention by the user, sometimes they do not.
No socket factory for 'https' protocol	<p>When a client fails with the following exception:</p> <pre>java.io.IOException: No socket factory for 'https' protocol at org.apache.axis.transport.http.HTTPSender.getSocket(HTTPSender.java:100) org.apache.axis.transport.http.HTTPSender.writeToSocket(HTTPSender.java:110) org.apache.axis.transport.http.HTTPSender.invoke(HTTPSender.java:120)</pre> <p>FIXME - it may have happened because...</p>

Error Code	Definition
No client transport named 'https' found	<p>When a client fails with the following exception:</p> <pre>No client transport named 'https' found at org.apache.axis.client.AxisClient.invoke(AxisClient.java:170) at org.apache.axis.client.Call.invokeEngine(Call.java:2726)</pre> <p>The client is most likely loading an incorrect <code>client-config.wsdd</code> configuration file.</p>
ConcurrentModificationException in Tomcat 5.0.x	<p>If the following exception is visible in the Tomcat logs at startup, it might cause the HTTPSValve to fail:</p> <pre>java.util.ConcurrentModificationException at java.util.HashMap\$HashIterator.nextEntry(HashMap.java:782) at java.util.HashMap\$EntryIterator.next(HashMap.java:824) at java.util.HashMap.putAllForCreate(HashMap.java:424) at java.util.HashMap.clone(HashMap.java:656) at mx4j.server.DefaultMBeanRepository.clone(DefaultMBeanRepository.</pre> <p>The HTTPSValve might fail with the following exception:</p> <pre>java.lang.NullPointerException at org.apache.coyote.tomcat5.CoyoteRequest.setAttribute(CoyoteRequestFacade.java:100) at org.apache.coyote.tomcat5.CoyoteRequestFacade.setAttribute(CoyoteRequestFacade.java:100) at org.globus.tomcat.coyote.valves.HTTPSValve.expose(HTTPSVAlve.java:100)</pre> <p>These exceptions will prevent the transport security from working properly in Tomcat.</p>
java.net.SocketException: Invalid argument or cannot assign requested address	<p>FIXME - what causes this?</p>
GAR deploy/undeploy fails with container is running error	<p>A GAR file can only be deployed or undeployed locally while the container is off. However, GAR deployment fail with this error even if the container is off. This usually happens if the container has crashed or was stopped from cleaning up its state files.</p>

3. Errors

Table 9.2. GRAM4 Errors

Error Code	Definition	Possible Solutions
<p>globusrun-ws - error querying job state</p>	<p>During job submission, an error like this occurs: globusrun-ws failed: Delegating user credentials...Done. Submitting job...Done. Job ID: xxxx Termination time: xxxx Current job state: Unsubmitted globusrun-ws: Error querying job state globus_soap_message_module: Failed sending request ManagedJobPortType_GetMultipleResourceProperties. globus_xio: An end of file occurred</p>	<p>Periodically, globusrun-ws will query the GRAM service to check on the job state. The "End of file" indicates that the GRAM server dropped a connection when globusrun-ws tried to read a response. This could be caused by temporary network issues between the client and service, or possibly caused by an overloaded service host.</p>
<p>globusrun-ws - error querying job state</p>	<p>During job submission, an error like this occurs: globusrun-ws failed: Delegating user credentials...Done. Submitting job...Done. Job ID: xxxx Termination time: xxxx Current job state: Unsubmitted globusrun-ws: Error querying job state globus_soap_message_module: Failed sending request ManagedJobPortType_GetMultipleResourceProperties. globus_xio: System error in read: Connection reset by peer globus_xio: A system call failed: Connection reset by peer</p>	<p>Periodically, globusrun-ws will query the GRAM service to check on the job state. The System error in read: Connection reset by peer indicates that the GRAM server dropped the connection while trying to write the response. This could be caused by temporary network issues between the client and service, or possibly caused by an overloaded service host.</p>

Error Code	Definition	Possible Solutions
glo- bus- run- ws - error sub- mit- ting job	During job submission, an error like this occurs: globusrun-ws -Ft PBS -F ht- tps://host.teragrid.org:8444 -submit -b -f /tmp/wsgram.rsl -o /tmp/wsgram.epr failed: Submitting job...Failed. globusrun-ws: Error submitting job globus_soap_message_module: Failed sending request ManagedJobFactoryPortType_createManagedJob. globus_xio: Operation was canceled globus_xio: Operation timed out	The Operation timed out indicates that the GRAM service was not able to accept the job request and respond in time. This could be caused by temporary network issues between the client and service, or possibly caused by an overloaded service host.

Chapter 10. Related Documentation

No related documentation links have been determined at this time.

Chapter 11. Internal Components

Internal Components¹

¹ [internal-components.html](#)

Glossary

C

Condor A job scheduler mechanism supported by GRAM. See <http://www.cs.wisc.edu/condor/> for more information.

J

job description Term used to describe a GRAM4 job for GT4.

L

LSF A job scheduler mechanism supported by GRAM.
For more information, see <http://www.platform.com/Products/Platform.LSF.Family/Platform.LSF/>⁷.

M

Managed Executable Job Service (MEJS) [FIXME]

Managed Job Factory Service (MJFS) [FIXME]

Managed Multi Job Service (MMJS) [FIXME]

multijob A job that is itself composed of several executable jobs; these are processed by the MMJS subjob.
See also [MMJS subjob](#)¹⁰.

P

Portable Batch System (PBS) A job scheduler mechanism supported by GRAM. For more information, see <http://www.openpbs.org>.

R

Resource Specification Language (RSL) Term used to describe a GRAM job for GT2 and GT3. (Note: This is not the same as RLS - the Replica Location Service)

⁷ <http://www.platform.com/Products/Platform.LSF.Family/Platform.LSF/>
¹⁰ #mmjs-subjob

S

scheduler

Term used to describe a job scheduler mechanism to which GRAM interfaces. It is a networked system for submitting, controlling, and monitoring the workload of batch jobs in one or more computers. The jobs or tasks are scheduled for execution at a time chosen by the subsystem according to an available policy and availability of resources. Popular job schedulers include Portable Batch System (PBS), Platform LSF, and IBM LoadLeveler.

W

Web Services Addressing
(WSA)

The WS-Addressing specification defines transport-neutral mechanisms to address web services and messages. Specifically, it defines XML elements to identify web service endpoints and to secure end-to-end endpoint identification in messages. See the W3C WS Addressing Working Group¹⁴ for details.

¹⁴ <http://www.w3.org/2002/ws/addr/>

Index

A

apis, 27
 links, 27
 overview, 27

C

compatibility, 2
containers, tested, 1

D

debugging, 38
 logging, 38
dependencies, 2

E

errors, 42, 45

F

features, 1

L

logging
 debugging, 38

P

platforms, tested, 1

R

resource properties, 31
 Managed Executable Job Port Type, 33
 Managed Job Factory Port Type, 31
 Managed Job Port Type, 32
 Managed Multi-Job Port Type, 35

S

services, 29

T

troubleshooting, 41
 check container log, 41
 check documentation, 41
 errors, 41
 mailing lists, 41

W

WSDL, 29