Ninf-G Version 4 Users Manual

This manual explains how to use Ninf-G Version 4 (installation, setup, and writing programs).

The reader is assumed to have knowledge and experience of the following.

- Some experience programming in C
- UNIX environment
- The make utility
- Globus Toolkit

This manual is comprised of the following components.

This section points out problems currently known in Ninf-G.

- Ninf-G Tutorial documents and sample programs which users can use to learn how to develop Ninf-G applications are included in doc/tutorial directory.
- Feedback on Ninf-G

  Should you encounter any problems using Ninf-G, please describe the problem and send it to the following mailing list.

  ninf-users@apgrid.org
- Feedback on this manual

  The Ninf development group is striving to improve this manual. Please send your comments or advice about the manual to the following e-mail address.

  ninf@apgrid.org

---

last update : $Date: 2007/07/24 08:36:20 $

1 Overview

1.1 GridRPC

Ninf-G uses the Globus Toolkit to provide an operating environment for GridRPC.

GridRPC is middleware that provides a model for access to remote libraries and parallel programming for tasks on a grid. Typical GridRPC middleware includes Ninf and Netsolve.

GridRPC is considered effective for use in the following cases.

- Commercial programs or libraries that use resources which are run on particular computers on the grid are sometimes provided only in binary format and cannot be executed on particular computers. There are also problems concerning licensing and source code compatibility. Furthermore, when using resources that can only be used with particular machines, such as video cameras, electron microscopes, telescopes and sensors, processing for the use of those resources on those machines is necessary.

  In such cases, an environment that allows the resources (including software) to be used on a particular computer is needed.

- When there are many programs that execute routines that do a large amount of computation on broadband servers on the grid, it takes a lot of time just to run parts of the program.

  The time required to run the program can be shortened by off-loading such program parts to a broadband server.

  In cases when there are strong demands on memory and disk space on the client machine so that broadband computation cannot be done, it is desirable to be able to do easily-understood offloading with no consideration given to argument marshalling.

- Execution of Parameter Sweep by multiple servers on the grid

  Parameter Sweep is a program that enables execution of computation on multiple servers in parallel, using some subset of the parameters. The respective servers run independently using different parameters, with virtually no dependence on other servers.

There are surprisingly many programs like Parameter Sweep.

The Monte Carlo method program is one of them.

Although Parameter Sweep can also be implemented with a Message Passing Interface (MPI), programming is rather simple with GridRPC and Parameter Sweep can be executed to match the (dynamically changing) scale of the grid (execution by multiple clusters, taking resource management, security, etc., into account).

- Ordinary or large-scale task parallel programs on a grid

  Task arrangement programs are easy to write with GridRPC. An API that supports the synchronization of various task arrangements with mixed exchange among multiple clients and servers can be used.

  GridRPC not only provides an interface for easy mathematical computation and scheduling of tasks for parallel execution, but the execution of processing that matches the (dynamically changing) scale of the grid is possible, as in the case of Parameter Sweep.

1.2 New features of Ninf-G Version 4

New features and functions have been added to Ninf-G Version 4 (Ninf-G4).

- Globus Toolkit Version 4 support

  Globus Toolkit Version 4 (GT4) provides a new framework and a new mechanism to provide job invocation (WS GRAM) and information services (Information Services: MDS4). Ninf-G4 supports the WS GRAM and MDS4 functions. The Ninf-G configuration file provides attributes to use these functions.

- Invoke Server

  Ninf-G4 has a new module called Invoke Server. This module enables support of many type of job invocation for Ninf-G. (WS GRAM, Pre-WS GRAM, UNICORE, ...)

  Any job submission interfaces can be used for remote process invocation by implementing Ninf-G Invoke Server for the interface. The detailed information on how to develop Ninf-G Invoke Server is described in Invoke Invoke Server Developer's Manual.

- Cascading RPC

  Ninf-G4 supports cascading RPC, which enables Ninf-G Executable to call GridRPC API. Cascading RPC implements hierarchical RPC which is a implementation technique to make applications scalable and to achieve high performance for fine-grained task parallel applications. Cascading RPC is available for Invoke Server GT4py by delegating full proxy certificates. The detailed information of this feature is available in Section 4.4.1. The other Invoke Servers such as Invoke Server SSH and Invoke Server Condor may be enabling cascading RPC, however they are not officially supported.

- Compatibility

  Ninf-G4 supports source code compatibility with Ninf-G Version 2 (Ninf-G2). Source codes of IDL and client programs are compatible between Ninf-G2 and Ninf-G4. Format of client configuration file in Ninf-G4 is expanded from Ninf-G2 and it is upper compatible with Ninf-G2.

  Ninf-G4 uses the same protocol with Ninf-G2 for communication between Ninf-G Client and Ninf-G Executable, thus mixed use of Ninf-G2 and Ninf-G4 is supported, i.e. Ninf-G2 client is able to call Ninf-G4 server and vice versa.

  In addition, GT2 functions (Pre-WS GRAM, Pre-WS MDS: MDS2) which are used by Ninf-G2 can be used by Ninf-G4 as well. Basically, Ninf-G2 users do not need to worry about compatibility problems. Only the user with interest in new Ninf-G4 features, must learn about the new capabilities of Ninf-G.

1.3 Overview of Ninf-G

Ninf-G is a set of library functions that provide an RPC capability in a Grid environment, based on the GridRPC API specifications.

1.3.1 Clients and servers

Ninf-G and the application programs that use Ninf-G consist of Ninf-G Executables that execute computation on server machines, and Ninf-G Clients that issue requests for computation to the Ninf-G Executables from client machines.

The Ninf-G Executables consist of functions that perform calculations (calculation functions) and a Ninf-G stub program that calls the calculation functions. Communication between clients and servers is accomplished by TCP/IP using a proprietary Ninf-G protocol.

The relationships between clients and servers are illustrated in Fig. 1.

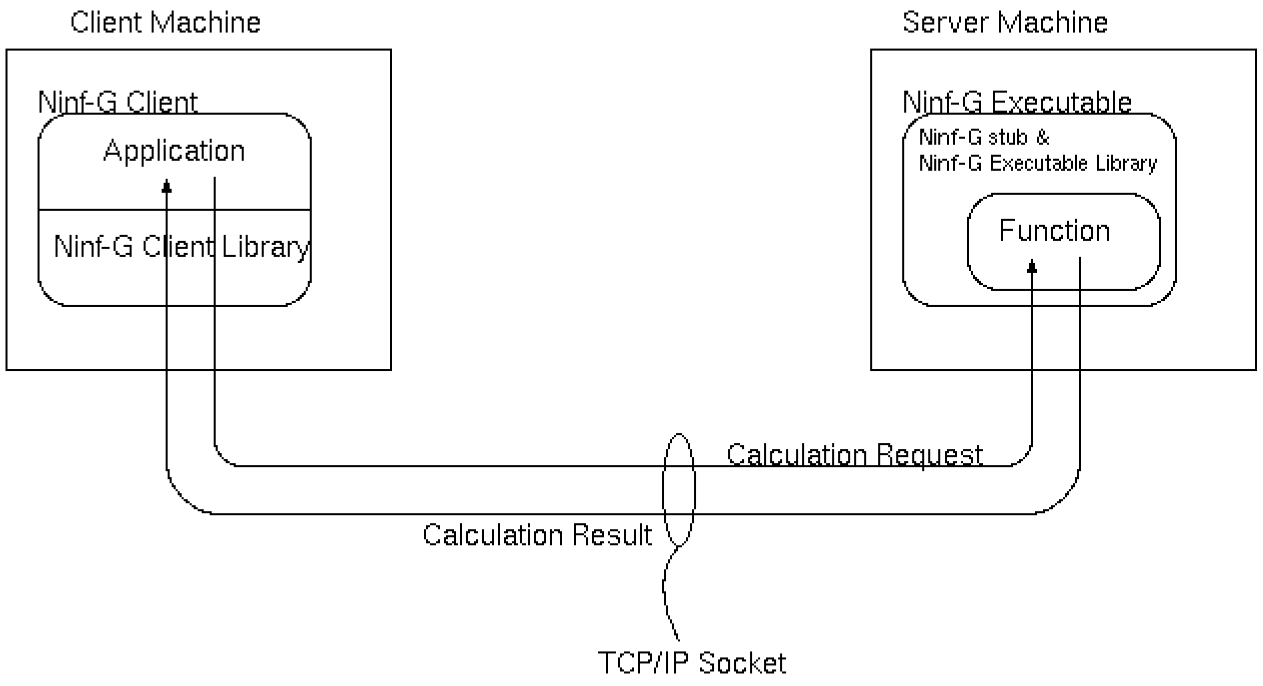

Figure 1: Clients and servers

1.3.2 Program hierarchy

Ninf-G employs the capabilities provided by the Globus Toolkit (http://www.globus.org/) for server machine authentication, information search, job start-up, communication and file transfer. The relations among applications, Ninf-G, the Globus Toolkit and the OS are illustrated in Fig. 2.

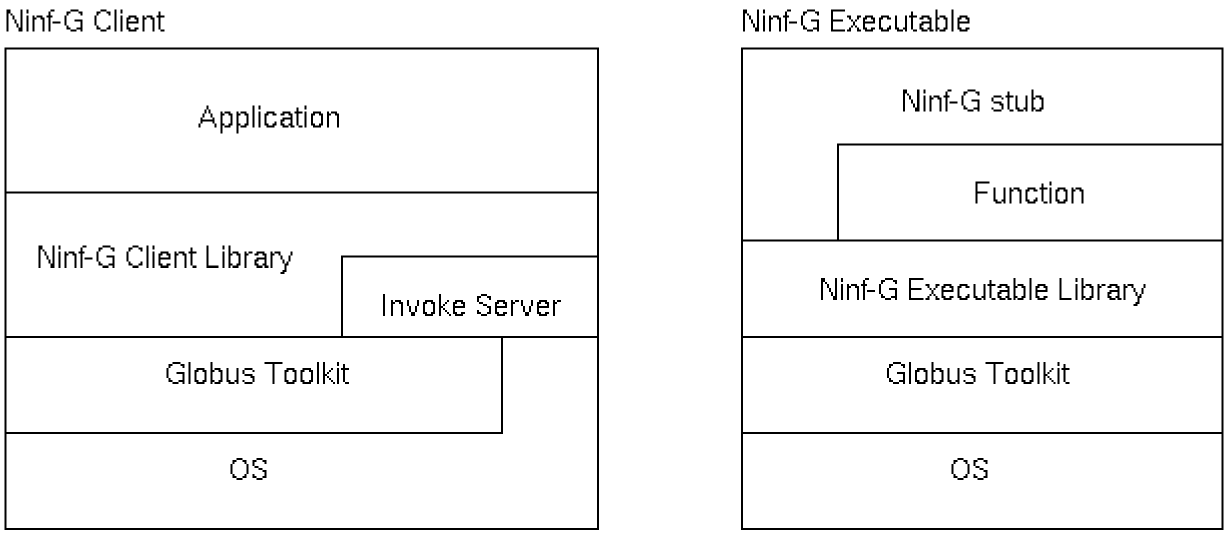

Figure 2: Program hierarchy

Ninf-G Clients are comprised of the following elements.

Applications	Programs written by Ninf-G users
Ninf-G Client Library	The set of API functions provided by Ninf-G for Ninf-G Clients
Globus Toolkit	The set of functions provided by the Globus Toolkit

Ninf-G Executables are comprised of the following elements.

Computation functions	Programs written by Ninf-G users
Ninf-G stub	A stub program produced by the Ninf-G stub generator
Ninf-G Executable Library	The set of API functions provided by Ninf-G for Ninf-G Executables
Globus Toolkit	The set of functions provided by the Globus Toolkit

1.3.3 Operating conditions

Ninf-G is supplied to the user as a source package, which includes the library functions (API) and utility commands. The operating environment required for the library functions and utility commands are shown in Table 1.

The usage of GT2 (implying the use of Pre-WS GRAM or MDS2) requires a GT2, GT3 or GT4 installation. Every Globus Toolkit has compatibility with GT2.

The usage of GT4 (implying the use of WS GRAM or MDS4) requires a GT4 installation.

Table1: Operating environment

Globus Toolkit	2.2 or later (2.4, 3.2, 4.0)
Python	2.3 or later
-	-
Target machine	SPARC
Operating system	Solaris 9 (SunOS 5.9)
Compiler	Sun Compiler or gcc 2.95
Globus Toolkit flavor	vendorcc32dbg, vendorcc32dbgpthr, gcc32dbg, gcc32dbgpthr

-	-
Target machine	PC-AT compatible (x86, Opteron)
Operating system	Linux(*1)
Compiler	gcc 2.95, gcc 3.0, 3.1, 3.2, 3.3, 3.4(*2)
Globus Toolkit flavor	gcc32dbg, gcc32dbgpthr, gcc64dbg, gcc64dbgpthr
-	-
Target machine	IBM Power4
Operating system	AIX 5.2
Compiler	C for AIX Compiler, Version 6
Globus Toolkit flavor	vendorcc32dbg or vendorcc32dbgpthr
-	-
Target machine	Apple Mac (PowerPC)
Operating system	MacOS X
Compiler	gcc 4.0.0
Globus Toolkit flavor	gcc32dbg or gcc32dbgpthr

(*1) We are checking operation with the following distributions.

- RedHat 8.0
- SuSE 8.1
- RedHat Advanced Workstation 2.1

(*2) There are problems with gcc 2.96, so we recommend you use gcc 2.95.x or gcc 3.0, 3.1, 3.2, 3.3, 3.4.

1.3.4 Requirements for operation

Ninf-G allows the definition of a single computation function (1) or multiple computation functions (2) for a Ninf-G Executable running on a server machine. The execution schemes for these are shown in Fig. 3. In either case, it is possible to execute just one computation function at a time on the Ninf-G Executable. To execute multiple computation functions at the same time, it is necessary to run multiple Ninf-G Executables. This is illustrated in Fig. 4.

In Ninf-G, the second scheme (2) is referred to as "Ninf-G Executable objectification" and the calling of the computation is referred to as a "method call."

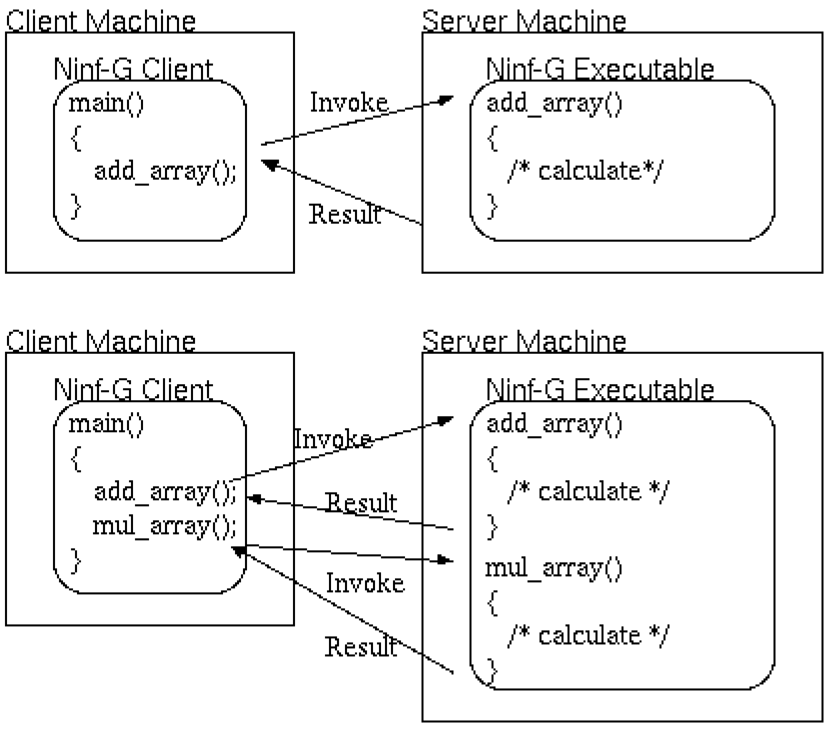

Figure 3: Overview of operation

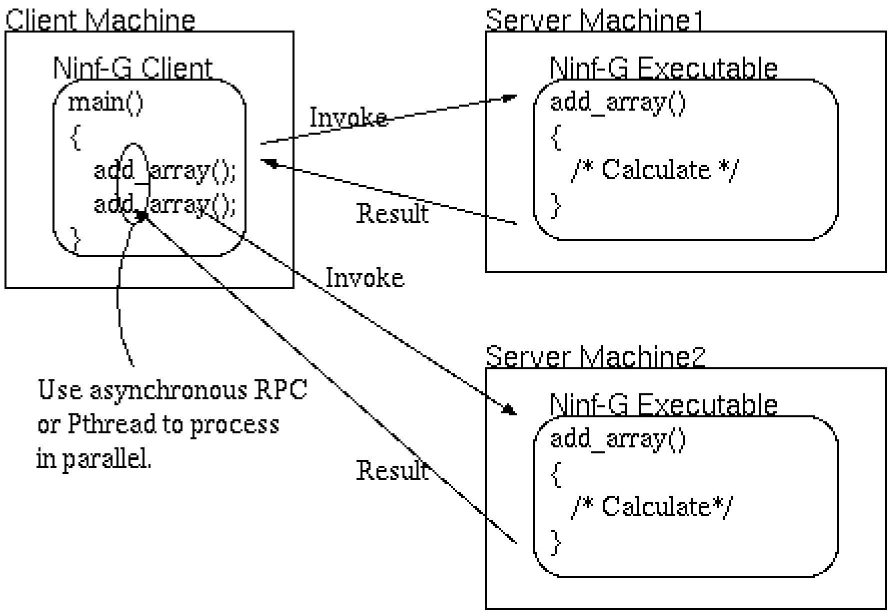

Figure 4: Parallel execution

Ninf-G provides handles for manipulating a Ninf-G Executable. Different handles are used for the two schemes, (1) and (2), described above. As shown in Table 2, two types of handles are provided, function handles and object handles.

Table 2: Handles

Function handle	Used for manipulation of a Ninf-G Executable for which a single function is defined
Object handle	Used for manipulation of a Ninf-G Executable for which multiple functions are defined

1.3.5 Starting up a Ninf-G Executable

Ninf-G Executables that run on server machines are started up from Ninf-G Clients, which run on client machines. A Ninf-G Executable is started up by performing the following procedure using the job control method provided by the Globus Toolkit or Invoke Server.

When running a Ninf-G Client program, however, there is no particular need for the user to be aware of this mechanism.

- The case which uses a Globus Toolkit Pre-WS GRAM directly, where Invoke Server is not used.

  1. A start-up request is sent from a Ninf-G Client to the gatekeeper, by Pre-WS GRAM.
  2. The gatekeeper starts up the jobmanager.
  3. The jobmanager starts up the Ninf-G Executable.

- The cases which use Invoke Server

  1. If the appropriate Invoke Server is not started on the Client, the Invoke Server process is started first.
  2. Ninf-G Client sends request to the Invoke Server to start the job.
  3. The Invoke Server invokes the Ninf-G Executable on the remote machine, by each Invoke Server individually.

  For example, if the Invoke Server for Globus Toolkit WS-GRAM is selected for use, the Invoke Server requests the remote WS-GRAM to perform the invocation. The requested remote WS-GRAM invokes the jobmanager, and the jobmanager invokes the Ninf-G Executable.

This process is shown in Fig. 5.

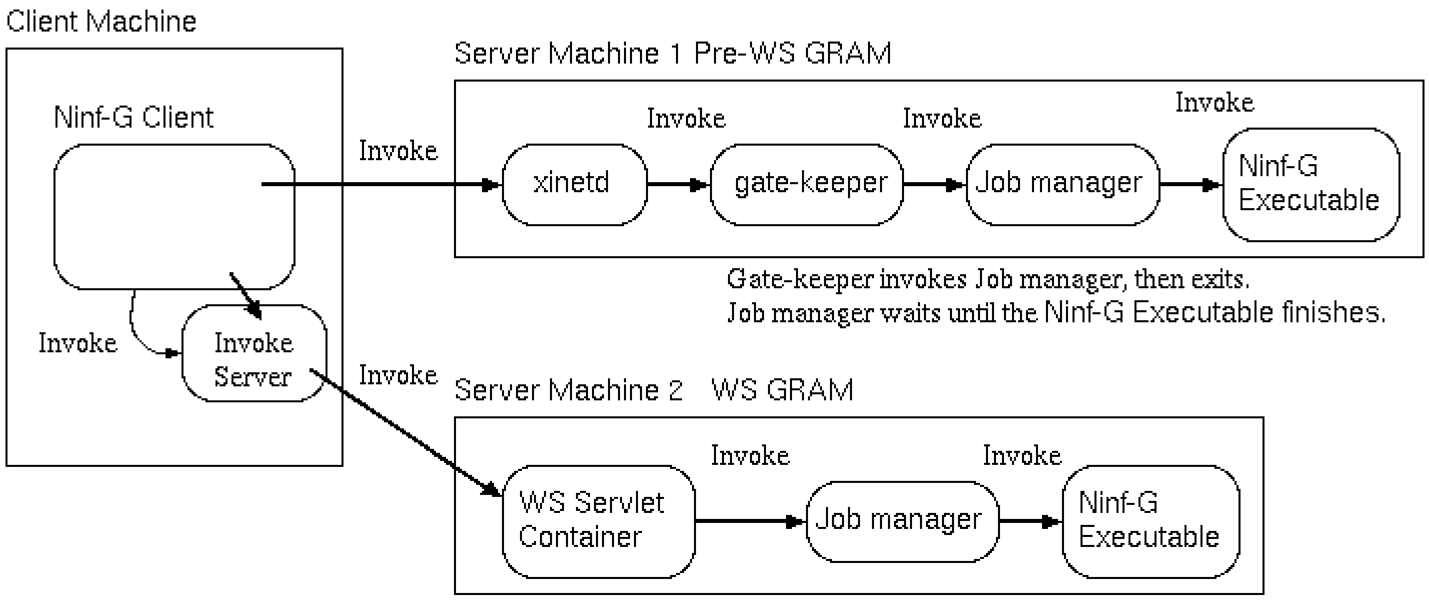

Figure 5: Starting up a Ninf-G Executable

1.3.6 Registering and accessing Ninf-G Executable information

Starting up a Ninf-G Executable requires path information that specifies the location of the Ninf-G Executable on that server machine. Information on the functions that are called by the Ninf-G Executable is also required. That information is collectively referred to as the Ninf-G Executable information. Ninf-G provides the following methods of registering and accessing Ninf-G Executable information.

When running a Ninf-G Client program, however, there is no particular need for the user to be aware of this mechanism.

- A file that contains the Ninf-G Executable information (a Local LDIF file) is placed on the client machine. The Ninf-G Client program obtains the Ninf-G Executable information from this Local LDIF file (Fig. 6).

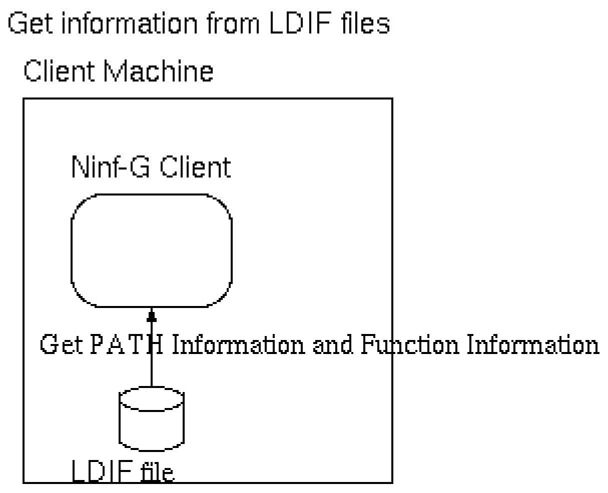
Get information from LDIF files
Client Machine
Ninf-G Client
Get PATH Information and Function Information
LDIF file

Figure 6: Local LDIF file

- The path information is defined in the configuration file for the Ninf-G Client on the client machine. The Ninf-G Client program obtains the path information from the configuration file and uses it to start up the Ninf-G Executable on the server machine. The function information is obtained from the Ninf-G Executable when it is started up (Fig. 7).

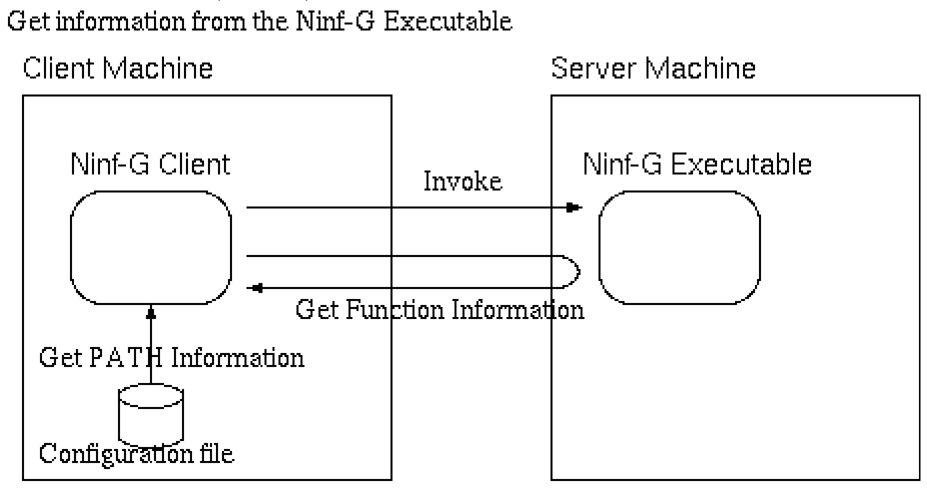

Figure 7: Ninf-G Executable

- The LDIF file is registered in the MDS(∗) of the server machine. The Ninf-G Client program obtains the Ninf-G Executable information from the MDS (Fig. 8).

  (∗) The information search function provided by the Globus Toolkit.

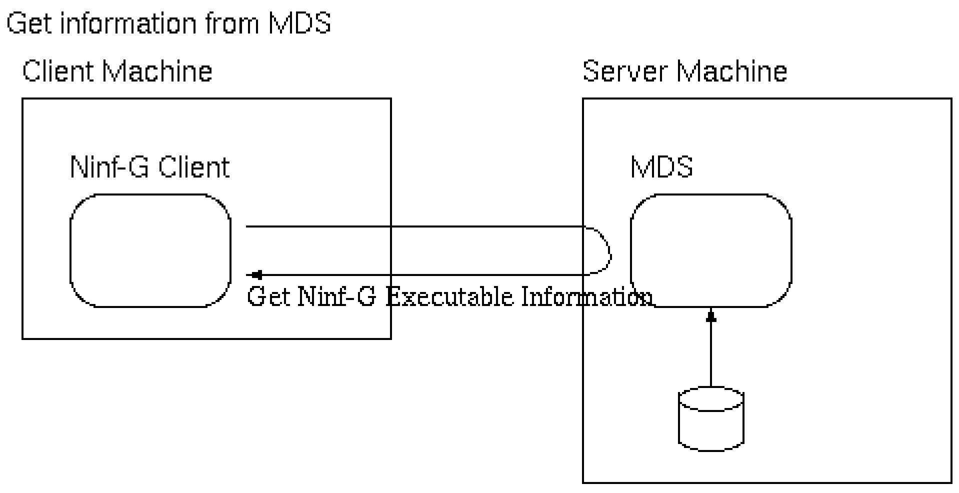

Figure 8: MDS

1.4 Definition of terms

Ninf-G Client

This is a program written by a user for the purpose of controlling the execution of computation. It is obtained by linking a user-written application program to the Ninf-G Client Library (and Globus Toolkit).

Ninf-G Client Library

The Ninf-G Client Library puts together the API used by application programs that run on client machines (Ninf-G Client API).

Ninf-G Executable

This is a program written for the execution of user requests for computation to be performed on a remote computer. It is obtained by linking a user-written computation function to stub code and the Ninf-G Executable Library (and Globus Toolkit). The stub code is produced by the stub generator according to the interface specifications of the user-defined computation function. The interface specifications are written in the Ninf-G IDL (Interface Description Language) specified by Ninf-G.

Ninf-G Executable Library

The Ninf-G Executable Library puts together the API (Ninf-G Executable API) used by a Ninf-G Executable.

Client (machine)

A machine that is running a Ninf-G Client.

Server (machine)

A machine that is running a Ninf-G Executable.

Function handle

A function handle is a data item whose type is grpc_function_handle_t. The function handle represents a mapping from a function name to an instance of that function on a particular server.

Object handle

An object handle is a data item whose type is grpc_object_handle_t_np. The object handle represents a mapping from a class name to an instance of that class on a particular server. The instance is called a Ninf-G remote object, and it is able to contain multiple methods.

Remote function

A computational function written by the user. (It might be only a single computation function for a Ninf-G Executable)

Remote method

A computational function written by the user. (It might be multiple computation functions for a Ninf-G Executable)

Session

A session extends from the time an RPC is made to the time its execution is completed.

In Ninf-G, a session extends

- from the time grpc_call() is called until the time it is completed (returns)
- from the time grpc_invoke() is called until the time it is completed (returns)
- from the time grpc_call_async()is called until the time grpc_wait*() is completed
- from the time grpc_invoke_async() is called until the time grpc_wait*() is completed.

GridRPC API

This is the standard API that systems implementing GridRPC should have. For the GridRPC C language API, standardization by the GGF WG is currently still in process.

Ninf-G IDL

IDL is the acronym for Interface Definition Language. It is a language for writing interfaces for the remote functions and remote methods defined by Ninf-G Executables.

Module name

This is the identifier for Ninf-G Executables. The user may specify any character string in the Ninf-G IDL.

1.5 Ninf-G Design

1.5.1 Reducing Overhead for Initialization of Function Handles

Ninf-G provides the following functionalities for reducing overhead for initialization of function handles.

- Creating multiple function handles via a single GRAM call and providing an API for utilizing the functionality.

  A single GRAM call usually takes several seconds for GSI authentication and a process invocation via the Globus jobmanager. This indicates that it will take more than several minutes to tens of minutes for hundreds of GRAM calls on a large-scale cluster. Also, many Globus jobmanager processes which will be launched on the front-end node will increase the load on the front-end node and cause the creation of additional overhead.

  Ninf-G implements a functionality which enables the creation of multiple function handles via a single GRAM call and provides an API for utilizing this functionality. For example, `grpc_function_handle_array_default_np()` takes three arguments, a pointer to an array of function handles, the number of function handles, and the name of the remote executable. When `grpc_function_handle_array_default_np()` is invoked, Ninf-G will construct an RSL in which the `count` attribute is specified as the number of function handles, and pass the RSL to the Globus GRAM. This allows invocation of multiple remote executables, i.e. initialization of multiple function handles, via a single GRAM call.

- Bypass MDS lookup for information retrieval

  Querying an MDS server for getting information on remote executables is a more difficult problem from a performance point of view, since it takes several minutes if the MDS server contains a large MDS tree. Although a useful resource discovery mechanism is essential for the acceptance of grid computing, we need to provide a practical scheme for information retrieval. Several approaches could be candidates for the implementation of information retrieval. For example, in CORBA, both a client and servers generate stubs and share information statically. Although this approach is straightforward and reduces the overhead for information retrieval, client programmers need to prepare IDL files for stub generation which constitutes a burden on client programmers. Ninf-G implements a functionality which enables it to retrieve the necessary information not from an MDS server, but from Local LDIF files which are placed on the client machine in advance. When Ninf-G Executables are generated on the server machine, the LDIF files are generated by the Ninf-G IDL compiler as well. The LDIF files should be copied to the client machine and can be specified in the client configuration file which is passed to the application as the first argument.

1.5.2 Making Data Transfers Efficient

Ninf-G provides the following functionalities for efficient data transfers and elimination of redundant data transfers.

- Implementation of a Ninf-G remote object

  Although the semantics of a remote executable is "stateless," it is desirable to provide a "stateful" remote executable since typical applications repeat computation for large data sets with different parameters. In the case of "stateless" executables, the executable needs to send the data in every remote library call, which would be a severe problem in a Grid environment. Ninf-G provides a "stateful" remote executable as a "Ninf-G remote object." A Ninf-G remote object can hold a "state" and be used to eliminate redundant data transfers between a client and servers. Ninf-G provides API functions such as `grpc_object_handle_init_np()` and `grpc_invoke_np()` for utilizing Ninf-G remote objects. `grpc_object_handle_init_np()` initializes a Ninf-G remote object and creates an object handle which is represents a connection between the client and the Ninf-G remote object. `grpc_invoke_np()` calls methods of the Ninf-G remote object as described in the Ninf-G IDL. A Ninf-G remote object is an instance of a class which is defined in an IDL file using `DefClass` statement on the server side. Multiple methods, which can be invoked by a client using a client API such as `grpc_invoke_np()`, can be defined in a class using the `DefMethod` statement.

- Compression of transferred data

  Ninf-G enables data transfers with compression. A flag which specifies whether to enable or disable data compression, and a data size as the threshold for compressing data can be specified in the client configuration file.

1.5.3 Compensating for the Heterogeneity and Unreliability of a Grid Environment

In order to compensate for the heterogeneity and unreliability of a Grid environment, Ninf-G provides the following functionalities:

- Client configuration formats for detailed description of server attributes

  The GridRPC API specifies that the first argument of a client program must be a "client configuration file" in which information required for running applications is described. In order to compensate for the heterogeneity and unreliability of a Grid environment, Ninf-G provides client configuration formats for detailed description of server attributes such as the Globus jobmanager, and a protocol for data transfers, etc.

- Timeout value for initialization of a function handle and RPC

  If a server machine is fully utilized, requests for initialization of function handles and remote library calls may be stuck in the queue and will not be launched for a long time, and this may cause deadlock of applications. Ninf-G provides a functionality to specify a timeout value for initialization of function handles as well as remote library calls. The timeout values can be specified in the client configuration file.

- Heartbeat

  A remote executable reports a heartbeat message to the client at a pre-specified interval. Ninf-G provides an API function for checking the heartbeat from the remote executable. The interval can be specified in the client configuration file.

- Client Callbacks

  Ninf-G provides a functionality called "client callbacks" by which a remote executable calls a function on the client machine. The client callback can be used for sharing status between the server and the client. For example, the client callback can be used for showing the interim status of computation at the client machine and in interactive processing.

- Cancellation of a session

Ninf-G provides a server-side API function named `grpc_is_canceled()` for checking the arrival of cancel requests from the client. If the client calls a `grpc_cancel()` function, `grpc_is_canceled()` returns `1`. In order to implement cancellation of a session, remote executables are required to call `grpc_is_canceled()` at an appropriate interval and return by itself, if `grpc_is_canceled()` returns `1`.

1.5.4 Supporting Debugging of Applications

Ninf-G provides functionalities which are useful for debugging. Ninf-G enables redirection of stdout and stderr of remote executables to the client machine. Log messages generated by Ninf-G and the Globus Toolkit can also be stored on the client machine. Furthermore, Ninf-G enables the launch of "gdb" on the server machine when a remote executable is launched on the server. These functionalities are made available by turning on the flags in the client configuration file.

1.6 Compatibility with Ninf-G2

- Source code compatibility

  The versions are source code compatible. Client-side application programs and server-side remote function programs that are used with Ninf-G2 can be used without modification.

- other compatibility

  The environment variable name, utility command name, and configuration file attribute names have compatibility with Ninf-G2.

1.7 Assumed environment for using Ninf-G

1.7.1 Prerequisites for installing Ninf-G

- If GT4 is used:
  1. GT4 Pre-WS MDS should be setup (using configure --enable-prewsmds option) if MDS2 is needed by the user.
  2. GT4 Information Services C bindings should be setup if MDS4 on C Client is needed by the user.
  3. The globus_core must be installed, which is not installed by binary installer. Source installer installs this module.
- If GT2 is used:
  1. All SDK bundles of the Globus Toolkit (resource, data, and information) must be built from source bundles.
  2. All SDK bundles of the Globus Toolkit (resource, data, and information) must have a common flavor by which those bundles are built. The flavor should be specified as the Globus flavor when installing (configuring) Ninf-G except when the flavor is gcc32dbg.
- Python 2.3 or later must be installed.

1.7.2 Environment variables for installing / using Ninf-G

- GPT_LOCATION must be set to the GPT installation directory (if GT3 or GT4 is used, GPT_LOCATION must be set as the same location as GLOBUS_LOCATION).
- GLOBUS_LOCATION must be set to the Globus Toolkit installation directory.
- Reading the user setting file for use by the Globus Toolkit, read ${GLOBUS_LOCATION}/etc/globus-user-env.{sh,csh}.
- NG_DIR must be set to the Ninf-G installation directory.
- Reading the user use environment setting file for use by Ninf-G, read ${NG_DIR}/etc/ninfg-user-env.{sh,csh}.
- NG_LOG_LEVEL specifies the loglevel that controls the produced error/warning messages during executions. This variable is set to 2 (Error) by sourcing ${NG_DIR}/etc/ninfg-user-env.{sh,csh}.

1.7.3 Execution Environment

- Ninf-G users must be capable of submitting jobs using the Globus Toolkit from a client machine on which Ninf-G Client programs will run to server machines on which the Globus gatekeeper is running and Ninf-G Executables will be launched by the Globus jobmanager.
- The server machines must be IP-reachable for the client machine, that is, the server machines should be capable of establishing a connection from the server machines to the client machine. This implies that the private IP address nodes can be utilized by Ninf-G if (1) the Globus gatekeeper is running on a gateway node, (2) a Globus jobmanager such as jobmanager-pbs is available for launching jobs on backend nodes, and (3) NAT is available on the gateway node so that backend nodes can connect to the client machine.

---

last update : $Date: 2008/09/29 03:21:51 $

2 Installation manual

Prerequisites for installing Ninf-G

- If GT4 is used:
  1. GT4 Pre-WS MDS should be setup (using configure --enable-prewsmds option) if MDS2 is needed by the user.
  2. GT4 Information Services C bindings should be setup if MDS4 on C Client is needed by the user.
  3. The globus_core must be installed, which is not installed by binary installer. Source installer installs this module.
- If GT2 is used:
  1. All SDK bundles of the Globus Toolkit (resource, data, and information) must be built from source bundles.
  2. All SDK bundles of the Globus Toolkit (resource, data, and information) must have a common flavor by which those bundles are built. The flavor should be specified as the Globus flavor when installing (configuring) Ninf-G except when the flavor is gcc32dbg.
- Python 2.3 or later must be installed.

Note: We recommend you use *pthr flavor. Some features don't work with non-*pthr flavor.

2.1 Downloading the Ninf-G package

Download the Ninf-G package from the download Web page

(http://ninf.apgrid.org/packages/welcome.html).

2.2 Creating a "ninf" user

It is recommended that you create a "ninf" user on the installation system.

It is also possible, however, to install Ninf-G in a location where the user has read and write privileges under general user privileges (that user's home directory, for example).

2.3 Installation

- Globus libraries built from source bundles must be installed on all nodes on which Ninf-G Clients and Executables may run. The libraries may be installed in a shared directory.
- Ninf-G libraries must be installed on the nodes on which Ninf-G Clients and Executables will be compiled. The libraries may be installed in a shared directory.
- The server_install command for registering the host information needs to be executed only on a front-end node on which the Globus gatekeeper is running.

- Expanding the source files

Move the files of the downloaded package to the directory in which the source files are to be expanded and then use the following command to expand the Ninf-G package files. (The 4.x.x in the command is the version number.)

```
% gunzip -c ng-4.x.x.tgz | tar xvf -
```

Executing the above command will create a "ng-4.x.x" directory and expand the Ninf-G source files in that directory.

Note: The Ninf-G package is created by GNU tar, so it requires you to use GNU tar to expand the package.

- Running the configure script

  (Move to the directory in which the source files are expanded.)

  ```
  % cd ng-4.x.x
  ```

  ```
  % ./configure
  ```

  Executing the above command, the host software environment is examined, and the execution environment for the tools used by compile is prepared (creating Makefile, etc.).

  Parameters such as those for specifying a particular Ninf-G installation directory can be included as shown below.

  ```
  % ./configure --prefix=/usr/local/ng --with-globusFlavor=gcc32dbgpthr
  ```

  In this example, the following parameters are specified.

  - The path to the Ninf-G installation directory
  - The flavor of Globus Toolkit to be used

  If you do not know about flavors, ask the administrator for flavors which the system-installed Globus Toolkit implies. (The flavors may be obtained by $GPT_LOCATION/sbin/gpt-query command instead.)

  If the Ninf-G user desires to use MDS2 or MDS4 on a C client, --with-mds2 or --with-mds4 options must be specified.

  Other options are described in section 2.4. The options that can be used with the configure command can be viewed with the following command.

  ```
  % ./configure --help
  ```

- Executing the make command

  Execute the following command in the directory for expanding the source files.

  ```
  % make
  ```

  Executing make generates the libraries needed by Ninf-G as well as an executable binary file.

  In Ninf-G, the Makefile is created by using the GNU autoconf configure script. If there is code in the Makefile written with the POSIX make program that cannot be executed, try using the GNU make program.

  To configure either the server environment only or the client environment only, run one of the following make commands.

  - To configure the server environment only

    The following command can be used to configure the server environment only.

    ```
    % make server
    ```

  - To configure the client environment only

    The following command can be used to configure the client environment only.

    ```
    % make client
    ```

- Installing the compiled files, etc.

  With owner privileges for the directory in which the files are to be installed (specified by a --prefix at the time of configure command; the default is "/usr/local"), execute the following command from the directory in which the source files were expanded.

  If you want to specify a number of CPUs for MPI for LocalLDIF(*.ngdef), you have to modify < package_dir > /etc/server.conf and change the variable of MPIRUN_NO_OF_CPUS.

  ```
  % make install
  ```

  Executing the above command copies the libraries and executable binaries created by executing the make command and the commands needed to run Ninf-G to the specified directory.

  To install either the server environment only or the client environment only, execute the make command as described below.

  - To install the server environment only

    It is possible to install only the server environment by executing the following command.

    ```
    % make install_server
    ```

  - To install the client environment only

    It is possible to install only the client environment by executing the following command.

    ```
    % make install_client
    ```

  (The following commands are executed only on the server machine (where the Ninf-G Executable is run).
  If MDS is not being used, the following tasks are not necessary. In that case, owner user privileges for $GLOBUS_LOCATION are also not needed.)

- Registering the host information

  If the Ninf-G Client uses MDS2 or MDS4, MDS setup is required. This is required only on gatekeeper nodes.

  Execute the following command with owner user privileges for $GLOBUS_LOCATION (for example, "globus").
  If you want to specify the number of CPUs for MPI, you have to edit server_install and change the variable of MPIRUN_NO_OF_CPUS.

  ```
  % cd ng-4.x.x/utility/script
  % ./server_install
  ```

- Settings for provision of Ninf-G Executable information by MDS2

  If the Ninf-G Client uses MDS2, MDS2 setup is required. This is required only on gatekeeper nodes.

  Execute the following items with owner user privileges for $GLOBUS_LOCATION (for example, "globus").

  - Add the following line to ${GLOBUS_LOCATION}/etc/grid-info-slapd.conf.

    ```
    "include ${GLOBUS_LOCATION}/etc/grid-info-resource.schema"
    ```

    Below that line, add the following line.

    ```
    "include ${GLOBUS_LOCATION}/etc/grpc.schema"
    ```

  - Restarting MDS

    ```
     % ${GLOBUS_LOCATION}/sbin/globus-mds stop
     % ${GLOBUS_LOCATION}/sbin/globus-mds start
    ```

- Settings for provision of Ninf-G Executable information by MDS4

  If the Ninf-G Client uses MDS4, MDS4 setup is required. This is required only on gatekeeper nodes.

Execute the following items with owner user privileges for $GLOBUS_LOCATION (for example, "globus").

- Building and deploying service

  ```
  % cd ng-4.x.x/infoService
  ```

  ```
  % ant
  ```

  ```
  % ant deploy
  ```

- Editing GrpcInfoService configuration

  Edit URL in wsa:Address element in $GLOBUS_LOCATION/etc/ng4grpcinfo/regist.xml . (You'll have to modify example.org to the IP-address or FQDN of your host.)

  Edit the hostName in $GLOBUS_LOCATION/etc/ng4grpcinfo/jndi-config.xml to match the following description.

  ```
  <parameter>
      <name>hostName</name>
      <value>example.org</value>
  </parameter>
  ```

  Note. This hostname is the name of the GRAM server host.

  Edit the infoDirPath in $GLOBUS_LOCATION/etc/ng4grpcinfo/jndi-config.xml to match the following description.

  ```
  <parameter>
      <name>infoDirPath</name>
      <value>/usr/local/gt4.0.0/var/gridrpc</value>
  </parameter>
  ```

  This may be $GLOBUS_LOCATION/var/gridrpc .

  Append following line to $GLOBUS_LOCATION/container-log4j.properties .

  ```
  log4j.category.org.apgrid=INFO
  ```

  Restart the WS Servlet container, and watch the console output. If the following message is output, registration was successful.

  ```
  ????-??-?? ??:??:??,??? INFO  impl.GrpcInfoHome [Thread-?,run:???] done rescan to regist
  ```

  Note: If you change the information of the executable in $GLOBUS_LOCATION/var/gridrpc, the following command forcibly updates this information immediately.

  ```
  % java -DGLOBUS_LOCATION=$GLOBUS_LOCATION ¥
   -classpath $GLOBUS_LOCATION/lib/ng4grpcinfo.jar:$CLASSPATH ¥
   org.apgrid.ninf.ng4.grpcinfo.client.RescanClient ¥
   -s https://[IP-ADDR or FQDN]:8443/wsrf/services/org/apgrid/ninf/ng4/grpcinfo/GrpcInfoService
  ```

2.4 Configure command options

The available options can be displayed with the following command.

```
% ./configure --help
```

The options that can be used with the configure script are described below.

- General configure command options

  The general configure command options have no effect on the Ninf-G functions.

General configure options

option	description
--cache-file=FILE	Cache test results in FILE
--help	Print help message
--no-create	Do not create output files
--quiet, --silent	Do not print `checking...' messages
--version	Print the version of autoconf that created configure

- Directory and file names

  Specify the location for installing Ninf-G.

  Specify the installation path for the targets listed below.

  The default values are shown.

Directory and file names

option	default	description
--prefix=PREFIX	/usr/local	Install architecture-independent files in PREFIX
--exec-prefix=EPREFIX	same as prefix	Install architecture-dependent files in EPREFIX
--bindir=DIR	EPREFIX/bin	User executables in DIR
--sbindir=DIR	EPREFIX/sbin	System admin executables in DIR
--libexecdir=DIR	EPREFIX/libexec	Program executables in DIR
--datadir=DIR	EPREFIX/share	Read-only architecture-independent data in DIR
--sysconfdir=DIR	EPREFIX/etc	Read-only single-machine data in DIR
--sharedstatedir=DIR	EPREFIX/com	Modifiable architecture-independent data in DIR
--localstate=DIR	EPREFIX/var	Modifiable single-machine data in DIR
--lib-dir=DIR	EPREFIX/lib	Object code libraries in DIR
--includedir=DIR	EPREFIX/include	C header files in DIR
--oldincludedir=DIR	/usr/include	C header files for non-gcc in DIR
--infodir=DIR	EPREFIX/info	Info documentation in DIR
--mandir=DIR	EPREFIX/man	Man documentation in DIR
--srcdir	configure dir or ...	Find the sources in DIR
--program-prefix=PREFIX		Prepend PREFIX to installed program names
--program-suffix=SUFFIX		Append SUFFIX to installed program names
--program-transform-name=PROGRAM		Run sed PROGRAM on installed program names

- Features and packages

Features and packages

option	**default**	**description**
--disable-FEATURE		Do not include FEATURE (same as --enable-FEATURE=no)
--enable-FEATURE[=ARG]	ARG=yes	Include FEATURE
--with-PACKAGE[=ARG]	ARG=yes	Use PACKAGE
--without-PACKAGE		Do not use PACKAGE (same as --with-PACKAGE=no)
--with-globusFlavor=FLAVOR	gcc32dbgpthr	Specify Globus runtime library flavor
--with-mds		Obsolete option (use mds2 or mds4)
--with-mds2	no	Include functions for getting information on servers and functions from PreWS MDS
--with-mds4	no	Include functions for getting information on servers and functions from WS MDS
--with-zlib	yes if available	Use zlib for compression
--with-largefile	yes	Support largefile
--with-cc=CC	cc	Specify C compiler to use
--with-opt=OPT		Specify C compiler options for optimization
--with-debug=OPT		Specify C compiler options for debuggable executable file creation

--with-cppflag=OPT		Specify C preprocessor options
--with-python		Specify python command path
--with-naregi	no	Support NAREGI SS.
--with-naregidir=NAREGIDIR	/usr/naregi	Specify the directory in which NAREGI Middleware has been installed.
--enable-gcc		Allow use of gcc if available
--enable-debug	no	Enable generate executable with debug symbol

appendix : Installing GT4

Ninf-G requires Globus Toolkit installation. This appendix shows how to accomplish GT4 installation. When you install GT4, refer to Globus Website for exact information.

This section gives hints for installing GT4.

a.1 Installing the Globus Toolkit

Make the temporary directory.

```
% mkdir dirForInstaller
```

Install GT4 (we recommend Version 4.0.1 or later (not 4.0.0) and "source" installer).

```
% cd dirForInstaller
% gunzip -c [TARBALL LOCATION]/gt4.0.1-all-source-installer.tar.gz | tar xf -
% cd gt4.0.1-all-source-installer
% ./configure --prefix=/path/to/gt4-install
% make
% make install
```

Note: Use of MDS2 on Ninf-G requires the --enable-prewsmds option on GT4.

See also information about setting the Globus Toolkit at following URL.

http://www.globus.org/toolkit/docs/4.0/admin/docbook/

Ninf-G4 requires some components of the Globus Toolkit. Following URLs provide information about setting of components.

- Security Configuration of WS container http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch06.html
- GridFTP (Ninf-G4 requires GridFTP server on the remote server) http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch08.html
- RFT http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch10.html
- WS-GRAM (GRAM4) http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch11.html

a.2 About Usage Statistics Collection by Globus Toolkit

Globus Toolkit provides Usage statistics. (See http://www.globus.org/toolkit/docs/4.0/Usage_Stats.html) If you desire to prevent this, match the following configuration changes.

Set the environment variable "GLOBUS_USAGE_OPTOUT" to "1."

- (csh, tcsh, etc)

  ```
  % setenv GLOBUS_USAGE_OPTOUT 1
  ```

- (sh, bash, etc)

  ```
  $ GLOBUS_USAGE_OPTOUT=1
  ```

  ```
  $ export GLOBUS_USAGE_OPTOUT
  ```

Comment out the "usageStatisticsTargets" parameter in the configuration file $GLOBUS_LOCATION/etc/globus_wsrf_core/server-config.wsdd

This setting,

```
    <globalConfiguration>
        <parameter name="usageStatisticsTargets"
                   value="usage-stats.globus.org:4810"/>
```

Delete or comment out as follows.

```
    <globalConfiguration>
        <!--parameter name="usageStatisticsTargets"
                   value="usage-stats.globus.org:4810"/-->
```

Also check the following URLs.

- http://www.globus.org/toolkit/docs/4.0/common/javawscore/admin-index.html#s-ja vawscore-Interface_Config_Frag-usageStatisticsTargets
- http://www.globus.org/toolkit/docs/4.0/data/gridftp/admin-index.html#s-gridftp-admin-usage

a.3 Setting up the environment

Setup the environment variable and execute the script for setting up the environment.

- (csh, tcsh, etc)

  ```
  % setenv GLOBUS_LOCATION /path/to/gt4-install
  % source $GLOBUS_LOCATION/etc/globus-user-env.csh
  ```

- (sh, bash, etc)

  ```
  $ GLOBUS_LOCATION=/path/to/gt4-install
  $ export GLOBUS_LOCATION
  $ . $GLOBUS_LOCATION/etc/globus-user-env.sh
  ```

a.4 Starting the Container

```
% cd $GLOBUS_LOCATION
% ./bin/globus-start-container
```

Starting the SOAP server at: https://[IP-ADDR]:8443/wsrf/services/ with the following services:

```
[1]: https://[IP-ADDR]:8443/wsrf/services/TriggerFactoryService
[2]: https://[IP-ADDR]:8443/wsrf/services/DelegationTestService
....
[48]: https://[IP-ADDR]:8443/wsrf/services/CASService
[49]: https://[IP-ADDR]:8443/wsrf/services/ManagedJobFactoryService
```

a.5 Testing the WS GRAM

Test the WS GRAM using following procedures.

```
% cd $GLOBUS_LOCATION
% ./bin/grid-proxy-init
  (input your passphrase)
% globusrun-ws -submit -job-description-file ¥
    $GLOBUS_LOCATION/test/globus_wsrf_gram_service_java_test_unit/test.xml
Submitting job...Done.
Job ID: uuid:[UUIDUUID-UUID-UUID-UUID-UUIDUUIDUUID]
Termination time: MM/DD/CCYY HH:MM GMT
Current job state: Unsubmitted
Current job state: Done
Destroying job...Done.
```

a.6 Installing Index Service Bindings

If the Ninf-G C Client uses MDS4, the Index Service Bindings should be installed with Globus Toolkit.

Execute the following items with owner user privileges for $GLOBUS_LOCATION (for example, "globus").

```
% cd gt4.0.1-all-source-installer
% make globus_c_wsrf_core_bindings-thr
% make globus_handler_ws_addressing-thr
% $GLOBUS_LOCATION/sbin/gpt-build index_service_bindings-1.2.tar.gz [flavor]
```

Note: threaded flavor recommended.

- How to build an index_service_bindings package

  An index_service_bindings package is not prepared at the beginning. The user must create the package.

  Note: For convenience, the Ninf-G4 package includes a generated index_service_bindings package on ng-4.x.x/external/index_service.

  1. Copy the non-thread flavor's template archive (the with-thread flavor version has problems).

     ```
     % cp ¥
     gt4.0.1-all-source-installer/source-trees/wsrf/c/parser/cgen/source/globus_wsrf_bindings_template.tar.gz ¥
     $GLOBUS_LOCATION/share/globus_c_wsrf_cgen
     ```

  2. Edit client-stub-source-doclit.tmpl.

     Edit and remove line 657 of $GLOBUS_LOCATION/share/globus_c_wsrf_cgen/client-stub-source-doclit.tmpl .

     ```
          646             if(chain)
          647             {
          648                 globus_handler_chain_register_invoke(
          649                     chain,
          650                     GLOBUS_HANDLER_TYPE_REQUEST,
          651                     request->client_handle->message,
          652                     $internal_func_prefix$_i_request_done_callback,
          653                     request);
          654             }
          655         }
          656
     !    657         globus_mutex_unlock(&request->mutex);  --- *** REMOVE THIS LINE ***
          658         break;
          659
          660     case $sname.toUpperCase()$_REQUEST_INVOKING_HANDLERS:
          661
          662         if(request->done)
          663         {
          664             if(request->result != GLOBUS_SUCCESS)
          665             {
          666                 result = request->result;
          667                 goto error_exit;
          668             }
     ```

     ```
     % $GLOBUS_LOCATION/bin/globus-wsrf-cgen -no-service ¥
      -s index_service -flavor gcc32dbg -d $PWD/bindings ¥
      $GLOBUS_LOCATION/share/schema/mds/index/index_service.wsdl
     ```

     The package is created in bindings/index_service_bindings-1.2.tar.gz

a.7 Installing GT4 by Binary installer

It is recommended to install Globus Toolkit from source installer. If Globus Toolkit is installed from binary installer, you need to install globus_core with the following command:

```
% $GLOBUS_LOCATION/sbin/gpt-build -nosrc <flavor>
```

Where flavor is the Globus Toolkit flavor you're passing to Ninf-G configure script.

(GT4 Admin Guide B.4. Using globus-makefile-header with a binary distribution)

a.8 Installed file composition

```
$GLOBUS_LOCATION/
+etc
| +grid-info-resource-ldif.conf (optional) (*1)
| +grpc.schema (optional)
| +gpt
| | +packages
| |   +ng4grpcinfo (optional)
| |     +undeploy.xml (optional)
```

```
| +ng4grpcinfo (optional)
|   +regist.xml (optional)
|   +server-config.wsdd (optional)
|   +jndi-config.xml (optional)
+lib
| +ng4grpcinfo.jar (optional)
| +ng4grpcinfo_stubs.jar (optional)
+share
| +schema
|   +ng4grpcinfo (optional)
|     +GrpcInfo_flattened.wsdl (optional)
|     +GrpcInfo.wsdl (optional)
|     +GrpcInfo_service.wsdl (optional)
|     +GrpcInfo_bindings.wsdl (optional)
+var
  +gridrpc (optional)
    +catldif (optional)
    +root.ldif (optional)
    +*.ldif (optional)

    *1) The information of Ninf-G was added to this file.

$NG_DIR
+bin
| +base64encode
| +globusflags
| +ng_cc
| +ng_delete_functions
| +ng_dump_functions
| +ng_gen
| +ng_gen_dif
| +ng_invoke_server.Condor (optional)
| +ng_invoke_server.GT2c
| +ng_invoke_server.GT4java (optional)
| +ng_invoke_server.GT4py
| +ng_invoke_server.SSH
| +ng_invoke_server.NAREGISS (optional)
| +ng_version
+doc
| +tutorial
| +users_manual
+etc
| +gpt-query-result.txt
| +ng_invoke_server.GTtempl
| +ninfg-user-env.csh
| +ninfg-user-env.sh
| +server.conf
+include
| +grpc.h
| +grpcError.h
| +grpcLocal.h
| +grpc_executable.h
| +net.h
| +ng.h
| +ngClientInternal.h
| +ngCommon.h
| +ngConfig.h
| +ngConfigFile.h
| +ngEx.h
| +ngExecutableInternal.h
| +ngFunctionInformation.h
| +ngInternal.h
| +ngPlatform.h
| +ngXML.h
+lib
  +classad.jar (optional)
  +condorAPI.jar (optional)
  +condorIS.jar (optional)
  +gt4invokeserver.py
  +gt4invokeserverconfig.py
  +ioutils.py
  +libexpat.a
  +libngclient.a
  +libngcommon.a
  +libngexecutable.a
  +libnggrpc.a
  +libngnet.a
  +libngutility.a
  +ng_invoke_server.GT4.py
  +ng_invoke_server.jar (optional)
  +ngisgt4.jar (optional)
  +ngutils.py
  +template.mk
  +template.sh
  +uuid.py
  +naregissIS.jar (optional)
  +naregiss_is_execute.sh (optional)
```

---

last update : $Date: 2008/09/12 08:27:42 $

3 Creating and setting up server-side programs

3.1 Creating a Ninf-G Executable

- Describing the interface information with Ninf-G IDL

  The following kinds of information are described with IDL.

    - Module name (Ninf-G Executable identifier)
    - Interface information
    - Other information required by Ninf-G Executable

  The following is a sample of IDL for implementing matrix multiplication. For detailed Ninf-G IDL specifications, see chapter 6, "Ninf-G IDL Specifications".

  Ninf-G IDL sample

```
Module mmul;

Define dmmul(IN int n, IN double A[n][n], IN double B[n][n],
                    OUT double C[n][n])
"... description ..."
Required "libxxx.o" /* specify library including this routine. */
Calls "C" mmul(n,A,B,C); /* Use C calling convention. */
```

  The following is an example of a "callback", that can be described with Ninf-G IDL.

  Ninf-G IDL sample (callback)

```
Module test;

Define callback_test(IN int a, OUT int *b,
                     callback_func(IN int c[1], OUT int d[1]))
{
    int executableStatus, clientStatus;
    executableStatus = calc(a, b);
    callback_func(executableStatus, &clientStatus);
    if (d == 1) {
        /* client is alive */
    }
}
```

  Note: You may find some compiler warnings when you compile Callback function, but they are harmless. For example, following compiler warning is output by Solaris/vendor-cc.

```
"warning: old-style declaration or incorrect type for: callback_test"
```

- Implementation of remote functions and remote methods

  The required remote functions and remote methods (those called in the IDL) must be implemented (in C or FORTRAN).

The source code (C) for the mmul() function used in the Ninf-G IDL sample shown above is presented below.

Remote function (Remote method) sample

```
void mmul(int n, double * a, double * b, double * c)
{
    double t;
    int i, j, k;

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            t = 0;
            for (k = 0; k < n; k++){
                t += a[i * n + k] * b[k * n + j];
            }
            c[i * n + j] = t;
        }
    }
}
```

Note: Implementing cancel processing

When implementing processing for canceling a session, call grpc_is_canceled_np() in the Ninf-G Executable Library API from within a remote function or remote method and check the return value. Coding for performing session cancel processing if the return value is 1 is needed.

Example of cancel processing implementation

```
#include <grpc_executable.h>

void  mmul(int n, double * a, double * b, double * c)
{
    double t;
    int i, j, k;

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            t = 0;
            for (k = 0; k < n; k++){
                t += a[i * n + k] * b[k * n + j];
            }
            c[i * n + j] = t;
        }

        if (grpc_is_canceled_np() == 1) {
            /* canceled */
            break;
        }
    }
}
```

- Using the Ninf-G generator to compile the IDL

  ```
  % ${NG_DIR}/bin/ng_gen
  ```

  Executing the above command generates the following.

  - Ninf-G stub for executing a remote function (remote method)
  - makefile for generating a Ninf-G Executable
  - LDIF file (for saving the interface information)
  - Local LDIF file (for saving the interface information)

- Creating the Ninf-G Executable

The makefile generated by the stub generator is used to execute the make command.

```
% make -f <module_name>.mak
```

An executable Ninf-G Executable is generated.

(The following tasks are not necessary if MDS is not being used. The Ninf-G Executable information can be acquired without MDS by using the Ninf-G Executable path information that is contained in the local LDIF file on the client side or in the Ninf-G Client configuration file.)

- Copying the LDIF file to the specified location

  The LDIF file generated by the stub generator is copied to $GLOBUS_LOCATION/var/gridrpc.

  ```
  % make -f <module_name>.mak install
  ```

  The content of the copied LDIF file is shown below.

  Example LDIF file

  ```
  dn: GridRPC-Funcname=mmul/dmmul, Mds-Software-deployment=GridRPC-Ninf-G,
  objectClass: GlobusSoftware
  objectClass: MdsSoftware
  objectClass: GridRPCEntry
  Mds-Software-deployment: GridRPC-Ninf-G
  GridRPC-Funcname: mmul/dmmul
  GridRPC-Module: mmul
  GridRPC-Entry: dmmul
  GridRPC-Path: /path/to/_stub_dmmul
  GridRPC-Stub:: ICA8Y2xhc3MgbmFtZT0ibW11bC9kbW11bCIgdmVyc2lvbj0iMS4wIiBud
   ...
  ```

  Note: Copying the LDIF file to the prescribed location as above enables provision of Ninf-G Executable information by MDS.

  Ninf-G provides utility commands (described below) for managing the Ninf-G Executable information supplied by MDS. Details on the commands can be found in chapter 8, "Utility Command Reference".

  - Display the Ninf-G Executable information provided by the MDS of the host. (the ng_dump_functions command)
  - Delete the Ninf-G Executable information from MDS (the ng_delete_functions command)

3.2 Setting up the Ninf-G Executable operating environment

- Preparing the configuration file for use by Ninf-G Executable

  Prepared according to the need for a configuration file for adjusting the location of the temporary files generated during filename type argument processing and for the log output.

  - System settings

    Written in ${GLOBUS_LOCATION}/var/gridrpc/ngstub.conf.

  - User settings

    Write ".ngstubrc" in the user's home directory.

  If the system settings and user settings described above both have the same setting items, the user settings have priority.

3.3 Specifications for the configuration file used by Ninf-G Executable

3.3.1 Structure of the configuration file

The Ninf-G Executable configuration file is a text file which contains the settings information that is required for operation of Ninf-G Executable.

- This file is placed on the server on which Ninf-G Executable runs.
- The system attribute file is located in $GLOBUS_LOCATION/var/gridrpc/. The file name is ngstub.conf.
- The user definition file (user settings file) is placed in the user's home directory. The file name is .ngstubrc.
- If the user wants to change a system attribute, the attribute to be changed is written in the user definition file.

An example of entries in the Ninf-G Executable configuration file is shown below.

```
#comment

attribute    value    # comment

attribute    value    # comment
attribute    value    # comment

...
```

- Each line consists of an attribute and its value.
- One line defines one attribute.
- What follows after a pound sign (#) is interpreted as a comment.
- The attribute and attribute value must be separated by one or more spaces or by a TAB character.
- Multiple attributes cannot be written on one line.
- The attribute and attribute value must be written on the same line.
- One attribute definition cannot extend over more than one line.
- An attribute must have a defining attribute value. An attribute alone defines nothing.
- Attribute values of attributes for which multiple definitions are possible cannot overlap.

The following description produces an error.

```
AttributeValue
        #  No delimiter between attribute and attribute value

Attribute    Value    Attribute    Value
        # Multiple attributes on a line

Attribute
Value
        #  attribute value extend across more than one line

Attribute
        #  No attribute value

tmp_dir /tmp
tmp_dir /var/tmp
        # Overlapping of the attribute values of an attribute
        #  that can not have multiple definitions
```

3.3.2 Attributes and attribute values

The attributes and their attribute value definitions are listed below.

```
tmp_dir                        Directory
loglevel                       [0-5]
loglevel_globusToolkit         [0-5]
loglevel_ninfgProtocol         [0-5]
```

```
loglevel_ninfgInternal        [0-5]
log_filePath                  File name
log_suffix                    Suffix
log_nFiles                    Number of files
log_maxFileSize               Number of bytes
log_overwriteDirectory        [true/false]
commLog_enable                [true/false]
commLog_filePath              PathFile name
commLog_suffix                Suffix
commLog_nFiles                FilesNumber of files
commLog_maxFileSize           FileSizeNumber of bytes
commLog_overwriteDirectory    [true/false]
save_stdout                   File name
save_stderr                   File name
handling_signals              Signals ...
continue_on_error             [true/false]
```

The definable attributes and attribute values are listed below.

Attribute	Attribute value	Default value	Multiple	Explanation
tmp_dir	For temporary files	/tmp	Not possible	The directory in which temporary files are placed Directory
loglevel	[0-5]	2	Not possible	Overall log level
loglevel_globusToolkit	[0-5]	2	Not possible	Error generated by the Globus Toolkit API
loglevel_ninfgProtocol	[0-5]	2	Not possible	Error concerning the Ninf-G protocol
loglevel_ninfgInternal	[0-5]	2	Not possible	Internal Ninf-G error
log_filePath	File name	stderr	Not possible	Log file name
log_suffix	Suffix	Sequence number	Not possible	Log file suffix
log_nFiles	Number of files	1	Not possible	Number of log output files
log_maxFileSize	Number of bytes	1M/unlimited	Not possible	Maximum number of bytes for log file
log_overwriteDirectory	[true/false]	False	Not possible	Over-write permission for log directory
commLog_enable	[true/false]	False	Not possible	Communication log output enabled/disabled
commLog_filePath	File name	stderr	Not possible	Communication log file name
commLog_suffix	Suffix	Sequence number	Not possible	Communication log file suffix
commLog_nFiles	Number of files	1	Not possible	Number of files for outputting communication log
commLog_maxFileSize	Number of bytes	1M/unlimited	Not possible	Maximum number of bytes for communication log file
commLog_overwriteDirectory	[true/false]	False	Not possible	Over-write communication log directory enabled/disabled Permitted

save_stdout	File name	None	Not possible	Save stdout to file
save_stderr	File name	None	Not possible	Save stderr to file
handling_signals	Signal names/numbers	SIGHUP SIGINT SIGTERM	Not possible	Handling signals
continue_on_error	[true/false]	False	Not possible	continue computation if an error occurs

The meanings of the log level values are described below.

Value	Meaning	Explanation
0	Off	Nothing is output.
1	Fatal	A fatal error is output.
2	Error	A nonfatal error is output.
3	Warning	A warning error is output.
4	Information	Guidance or other such information is output.
5	Debug	Debugging information is output.

- tmp_dir (temporary file directory)

  The directory in which temporal files are placed for passing the filename type arguments to a remote method.

  When omitted, if TMPDIR environment variable is defined, it is used; otherwise, "/tmp" is used.

- loglevel* (log level)

  The log level is specified for all log categories by loglevel and for each category individually by loglevel_*.

  When the log level for each category has not been specified, the log level for all categories is applied.

  When omitted, the value of the NG_LOG_LEVEL environment variable is used.
  If the NG_LOG_LEVEL environment variable is not set, 2 (Error) is used as the default value of loglevel.

- log_filePath (log file name)

  The name of the file to which the log is output is specified in the log file name.

  The file name may include a path that includes a directory (e.g., "/home/myHome/var/logFile").

  The file and directory name can include the following specifiers.

  - "%t"

    "%t" is replaced with the date as year, month and day, and the time in hours, minutes, seconds and milliseconds ("yyyymmdd-hhmmss-MMM") (e.g., "/ home/myHome/var/logDir%t/logFile" is replaced by "/home/myHome/var/logDir20030101-154801-123/logFile").

  - "%h"

    "%h" is replaced with the Ninf-G Executable hostname.

  - "%p"

    - "%p" is replaced with the process id of the Ninf-G Executable.

  The Ninf-G Executable id number is added to the end of the file name.

  When omitted, the log is output to standard error. If the log file name is omitted, the log_suffix, log_nFiles, and log_maxFileSize are ignored.

  Note: When the Ninf-G Executable exits abnormally on startup, the executable id is not added and the hostname and process id are added to the end of the file name.

- log_suffix (log file suffix)

  When a log file is specified, this specifies the suffix used when the log file is created.

  If a suffix is specified, the generated file name will be from "filename[000].suffix" to "filename[nnn].suffix". If omitted, the generated file name will be from "filename.[000]" to "filename.[nnn]". The number of files minus 1 is "nnn."

  The number of digits in "nnn" is the same as the number of digits in number of files minus 1. For example, if the number of files is set to 100, then the number will range from "00" to "99."

- log_nFiles (the number of files for log output)

  This is the number of files created for log output.

  0 indicates that an unlimited number of files can be output. A negative value results in an error.

  If omitted, the value 1 is used.

- log_maxFileSize (maximum number of bytes for the log file)

  This is the maximum number of bytes for the log file. A unit indicator from among "kKmMgG" can be appended to the numerical value to indicate Kilobytes (1024 bytes), Megabytes (1024 Kbytes), or Gigabytes (1024 Mbytes).

  If omitted, the value will be unlimited if the number of files is one, or 1 Mbyte if the number of files is two or more.

- log_overwriteDirectory (over-write permission for the directory in which the log files are generated)

  This establishes overwrite permission for the directory. If the specified directory exists, this specifies whether the creation of log files in that directory is enabled or disabled.

  Operation in the case that the directory exists is shown below.

    - true: If the file specified by log_filePath exists in the directory, that file is overwritten.
    - false: Error.

- commLog_enable (whether communication log output is enabled or disabled)

  This specifies whether the communication log output function is enabled or disabled. If 'true' is specified, the communication log is output. If not specified, the default value is false.

- commLog_filePath (communication log file name)

  The name of the file to which the communication log is output is specified in the log file name.

  The file name may include a path that includes a directory (e.g., "/home/myHome/var/logFile").

  The file and directory name can include the following specifiers.

    - "%t"

    "%t" is replaced with the date as year, month and day, and the time in hours, minutes, seconds and milliseconds ("yyyymmdd-hhmmss-MMM") (e.g., "/ home/myHome/var/logDir%t/logFile" is replaced by "/home/myHome/var/logDir20030101-154801-123/logFile").

  - "%h"

    "%h" is replaced with the Ninf-G Executable hostname.

  - "%p"

    "%p" is replaced with the process id of the Ninf-G Executable.

  The Ninf-G Executable id number is added to the end of the file name.

  When omitted, the log is output to standard error. If the communication log file name is omitted, the commLog_suffix, commLog_nFiles, and commLog_maxFileSize are ignored.

- commLog_suffix (communication log file suffix)

  When the communication log file is specified, this specifies the suffix used when the log file is created.

  If a suffix is specified, the generated file name will be from "filename[000].suffix" to "filename[nnn].suffix". If omitted, the generated file name will be from "filename.[000]" to "filename.[nnn]". The number of files minus 1 is "nnn."

  The number of digits in "nnn" is the same as the number of digits in the number of files minus 1. For example, if the number of files is set to 100, then the number will range from "00" to "99."

- commLog_nFiles (number of files for communication log output)

  This is the number of files created for communication log output.

  0 indicates an unlimited number of files can be output. A negative value results in an error.

  If omitted, the value 1 is used.

- commLog_maxFileSize (maximum number of bytes for the communication log file)

  This specifies the maximum number of bytes for the communication log file. A unit indicator from among "kKmMgG" can be appended to the numerical value to indicate Kilobytes (1024 bytes), Megabytes (1024 Kbytes), or Gigabytes (1024 Mbytes).

  If omitted, the value is either unlimited if the number of files is one or 1 Mbyte if the number of files is two or more.

- commLog_overwriteDirectory (over-write permission for the directory in which the communication log files are generated)

  This establishes overwrite permission for the directory. If the specified directory exists, this specifies whether the creation of log files in that directory is enabled or disabled. Operation in the case that the directory exists is shown below.

  - true: If the file specified by log_filePath exists in the directory, that file is overwritten.
  - false: Error.

- save_stdout (Save stdout to file)

  This specifies the file name to save stdout.

  If this attribute is set, stdout is saved to the specified file.

  If the given file name for both save_stdout and save_stderr attribute is the same, output is shared to one file in arbitrary order.

The output file is opend by append mode.

If omitted, stdout output is delivered to Ninf-G Client or discarded.

- save_stderr (Save stderr to file)

  This specifies the file name to save stderr.

  If this attribute is set, stderr is saved to the specified file.

  If the given file name for both save_stdout and save_stderr attribute is the same, output is shared to one file in arbitrary order.

  The output file is opend by append mode.

  If omitted, stderr output is delivered to Ninf-G Client or discarded.

- handling_signals

  This attribute specifies signals which will be caught by Ninf-G Executable.

  When the Ninf-G Executable catches the signal, Ninf-G cleans up all temporary files, and exits. This clean up process is performed only for signals which are specified in this attribute.

  The signals are specified by either signal name or signal number. Multiple signals can be specified by space-delimited enumeration. The value "none" can be specified if no signals need to be caught.

  If ommitted, SIGINT, SIGTERM and SIGHUP will be caught by Ninf-G Executable.

  Note: This attribute is available for Ninf-G Version 4.2.0 or later.

- continue_on_error (continue computation if an error occurs)

  This attribute is used to control behaviors of a Ninf-G Executable when a communication error occurs. Such a communication includes explicit one (e.g. data transfer) as well as implicit one (e.g. heartbeating).

  If the value is set to 'false', which is the default value, the Ninf-G Executable immediately exits from its execution when a communication error occurs. This enables Ninf-G Executables to release computing resources immediately after the error. It is also useful to avoid Ninf-G Executables remaining as zombie processes when the Ninf-G Client would die.

  If the value is set to 'true', a Ninf-G Executable does not exit and continues to run until the callee function or method will be completed. This configuration is valuable if the callee functions and methods record results of the computation as files in the server machines. In this case, it may be worth to continue the execution of the Ninf-G Executable even if a communication error occurs.

  If omitted, the value 'false' is used.

  Note: Even if the value is set to true, Ninf-G Executables may exit by catching signals from queueing system like SGE or PBS, when the Ninf-G Client was killed by SIGHUP, SIGINT or SIGTERM signal.

  Note: In order for Ninf-G Executables to detect an error immediately when the connection to the Ninf-G Client was closed, Ninf-G Executables must be linked with the pthread flavor of the Globus Toolkit libraries.

---

last update : $Date: 2008/03/28 06:23:30 $

4 Creating and setting up client-side programs

4.1 Creating a Ninf-G Client

- Creating an application program

  Programs for executing remote functions and remote methods are written using the GridRPC API. The creation of these programs is described in section 4.6.

- Setting the NG_COMPILER environment variable

  (If the application program is written in C, there is no particular need for the NG_COMPILER settings.)

  The NG_COMPILER environment variable is used to specify the compiler to be used to compile the Ninf-G Client. The default compiler for ng_cc is cc. Required options can be specified in NG_COMPILER in addition to the compiler name (path). When g++ is used, for example, NG_COMPILER is set in the following way.

  ```
  'g++ -Wall -g'
  ```

- Setting the NG_LINKER environment variable

  (If the application program is written in C, there is no particular need for the NG_LINKER settings.)

  The NG_ LINKER environment variable is used to specify the linker to be used to link the Ninf-G Client. The default linker for ng_cc is cc. Required options can be specified in NG_ LINKER in addition to the linker name (path). When g++ is used, for example, NG_ LINKER is set in the following way.

  ```
  'g++ -Wall -g'
  ```

- Creating a Ninf-G Client program

The application program created earlier is compiled using the ng_cc command, thus creating a Ninf-G Client program. An example of using the ng_cc command is shown below.

```
% ${NG_DIR}/bin/ng_cc -g -o test_client test_client.c

test_client.c:  Ninf-G Client source program
test_client:    Ninf-G Client executable program
                (The results of compiling the source code.)
```

4.2 Setting up the Ninf-G Client operating environment

- Preparing the Ninf-G Client configuration file

  This is the configuration file for settings concerning servers, clients, and the MDS.

  (The configuration file specifications are described in section 4.3.)

- Preparing the files specified by the configuration file

  Prepare the following files as specified in the configuration file.

  - The configuration file specified by INCLUDE section
  - The local LDIF file specified by LOCAL_LDIF section
  - The Ninf-G Executable executable file for which the staging is specified by FUNCTION_INFO section

  Note: The local LDIF file is generated when the Ninf-G Executable is created on the server. When used, the local LDIF file that is generated on the server that executes the remote functions or remote methods must be copied to a place where it can be used from the Ninf-G Client.

  Note: The environment variable LD_LIBRARY_PATH must be specified appropriately in the client configuration file if the Ninf-G Executable file will be staged to a system whose software environment is different from the build environment of the Ninf-G Executable.

4.3 Ninf-G Client configuration file specifications

4.3.1 Structure of the configuration file

The Ninf-G Client configuration file is a text file which contains settings information that is required for the operation of a Ninf-G Client. It consists of seven sections, which are INCLUDE, CLIENT, LOCAL_LDIF, FUNCTION_INFO, MDS_SERVER, INVOKE_SERVER, SERVER, and SERVER_DEFAULT.

- The INCLUDE section describes the files to be included.
- The CLIENT section describes information related to the Ninf-G Client.
- The LOCAL_LDIF section describes information for the local LDIF file.
- The FUNCTION_INFO section describes information on the remote function.
- The MDS_SERVER section describes information concerning the MDS server.
- The INVOKE_SERVER section describes information concerning the Invoke Server.
- The SERVER section describes information concerning Ninf-G Executable.
- The SERVER_DEFAULT section describes the default values for the SERVER section.

Examples of section and attribute descriptions are shown below.

```
#comment
<section>
attribute    value    # comment

attribute    value    # comment
attribute    value    # comment
</section>
```

```
...
```

- The definitions of each section begin with <section > and end with </section >.
- <section> and </section> are called section tags, or simply tags.
- Anything following a pound sign (#) is regarded as a comment.
- The information described in each section includes attributes and attribute values.
- There cannot be multiple descriptions in a section that does not allow multiple definitions.
- In a section that does not allow multiple definitions, the key attribute cannot have redundant attribute values.
- A section tag and an attribute cannot be written on the same line.
- At least one space or one tab character must separate an attribute and its attribute value.
- Multiple attributes cannot be defined on one line.
- An attribute and its attribute value must be written on one line.
- The backslash character (¥) cannot be used to extend a definition over multiple lines.
- Attributes which can have multiple definitions cannot have redundant attribute values.
- Upper case and lower case letters are distinguished, so the attribute name 'aaa', for example, cannot be written as Aaa or AAA, etc. The attribute values 'aaa' and 'AAA' are also judged as different.

Examples of errors in description are shown below.

```
<section>Attribute Value
                             # Section and attribute on the same line

AttributeValue     # No delimiter between the attribute and attribute value

Attribute  Value   Attribute  Value
                             # Multiple attributes on the same line

Attribute                    # Attribute and attribute value
Value                        # extend across more than one line.

Attribute ¥
Value                        # Line continued with a backslash (¥)

Attribute                    #  No attribute value
</section>

<CLIENT>                     # Multiple definitions in a section where
...                          #
</CLIENT>                    #
<CLIENT>                     # multiple definitions are not allowed
...                          #
</CLIENT>                    #

<MDS_SERVER>                 # Attribute value redundancy within a section
hostname  example.org        #  where multiple definitions are allowed
</MDS_SERVER>
<MDS_SERVER>
hostname  example.org        #
</MDS_SERVER>                #

<LOCAL_LDIF>                 # Attribute value redundancy
filename  example.ngdef      # for an attribute that allows
filename  example.ngdef      #  multiple attribute values
</LOCAL_LDIF>

<SERVER>
HostName  example.com        # Upper and lower case letters
hostname  EXAMPLE.COM        # Upper and lower case letters
</SERVER>
```

4.3.2 Specifying the unit for time

When a time value is specified as an attribute value (for time-out or other such attributes), a unit of time can be specified (e.g., 30 s, 30 sec or 30 seconds).

It is also possible to specify 'minute' or 'hour' as the time unit. Character strings such as 'se' and 'seco' are also interpreted to mean second, but strings that are not contained in 'second', such as 'set' will cause an error.

4.3.3 Specifying the unit for number of bytes

When specifying the number of bytes as an attribute value for attributes such as log file size, units (*) such as 1 K or 1 Kilo can be specified. Mega and Giga can also be specified as a unit for number of bytes.

Character strings such as Me, Meg, etc. are also interpreted to mean Mega, but strings that are not contained in Mega, such as Ma cause an error.

(*) 1 K = 1024 bytes, 1 M = 1024 Kbytes, 1 G = 1024 Mbytes

Each section of the configuration file is described below.

4.3.4 INCLUDE section

The INCLUDE section allows multiple definitions.

An example of an INCLUDE section description is shown below.

```
<INCLUDE>
filename    file name
filename    file name
</INCLUDE>
```

The attributes and attribute values of the INCLUDE section are shown below.

Attribute	Attribute value	Default value	Multiple	Explanation
filename	File name	None	Yes	File to be included

- filename (file to include)

  The file name of the configuration file to be read is specified. The file to be read must be in the configuration file format.

4.3.5 CLIENT section

The CLIENT section does not allow multiple definitions.

An example of a CLIENT section description is shown below.

```
<CLIENT>
hostname                    Host name
save_sessionInfo            Session information count
loglevel                    [0-5]
loglevel_globusToolkit      [0-5]
loglevel_ninfgProtocol      [0-5]
loglevel_ninfgInternal      [0-5]
loglevel_ninfgGrpc          [0-5]
log_filePath                File name
log_suffix                  Suffix
log_nFiles                  Number of files
log_maxFileSize             Number of bytes
log_overwriteDirectory      [true/false]
tmp_dir                     Directory
refresh_credential          Seconds
invoke_server_log           log name
fortran_compatible          [true/false]
handling_signals            Signals ...
listen_port                 Port number
```

```
listen_port_authonly       Port number
listen_port_GSI            Port number
listen_port_SSL            Port number
tcp_nodelay                [true/false]
</CLIENT>
```

The attributes and attribute values of the CLIENT section are shown below.

Attribute	Attribute value	Default value	Multiple	Explanation
hostname	Host name	globus_libc_ gethostname()	No	The host name of the client
save_sessionInfo	Session information count	256	No	The number of session information units to be saved
loglevel	[0-5]	2	No	Overall log level
loglevel_globusToolkit	[0-5]	2	No	Globus Toolkit API error log level
loglevel_ninfgProtocol	[0-5]	2	No	Ninf-G protocol error log level
loglevel_ninfgInternal	[0-5]	2	No	Ninf-G internal error log level
loglevel_ninfgGrpc	[0-5]	2	No	Grid RPC API error log level
log_filePath	File name	stderr	No	The log file name
log_suffix	Suffix	Sequence number	No	The log file suffix
log_nFiles	Number of files	1	No	The number of files for log
log_maxFileSize	Number of bytes	1M/unlimited	No	Maximum number of bytes for log file
log_overwriteDirectory	[true/false]	false	No	Over-write permission for log directory
tmp_dir	For temporary files	/tmp	No	The directory in which temporary files are placed
refresh_credential	Seconds	0	No	Refreshing proxy credential interval
invoke_server_log	log name	None	No	Invoke Server log filename
fortran_compatible	[true/false]	false	No	Fortran compatible mode
handling_signals	Signal names/numbers	SIGHUP SIGINT SIGTERM	No	Handling signals
listen_port	Port number	0	No	The port number for listening requests for unencrypted connections
listen_port_authonly	Port number	0	No	The port number for listening requests for connections by authentication only
listen_port_GSI	Port number	0	No	The port number for listening requests for connections encrypted by GSI
listen_port_SSL	Port number	0	No	The port number for listening requests for connections encrypted by SSL.
tcp_nodelay	[true/false]	false	No	TCP_NODELAY socket option

The log level can be specified by using the strings listed in the 'Meaning' column in the table below as well as by using a number value.

The meanings of the log level values are described below.

Value	Meaning	Explanation
0	Off	Nothing is output.
1	Fatal	A fatal error is output.
2	Error	A nonfatal error is output.
3	Warning	A warning error is output.
4	Information	Guidance or other such information is output.
5	Debug	Debugging information is output.

- hostname (client host name)

  This host name is the host name of the client machine on which the Ninf-G Client is running. It is used by the Ninf-G Executable when connecting to a Ninf-G Client.

  The host name can be specified by an IP address such as 192.168.0.1 as well as by the ordinary host name.

  If omitted, the hostname returned by the Globus Toolkit API globus_libc_gethostname() function is used. The hostname is equivalent to the output of the globus-hostname command.

  The main purposes for which this is used are described below.

  - When the Ninf-G Executable uses the DNS (Domain Name System), but for some reason the IP address cannot be resolved from the host name of the client machine on which the Ninf-G Client is running.
  - The host name set for the client machine on which the Ninf-G Client is running and the host name derived in reverse from the IP address using DNS are different.

  Note: If the hostname attribute is set, the environment variable GLOBUS_HOSTNAME is overwritten by its value.

  Note: The hostname attribute may not be set appropriately when globus_libc_gethostname() is called before grpc_initialize().

- save_sessionInfo (number of session information units to save)

  This is the number of session information units stored internally by Ninf-G.

  If the number defined here is exceeded, the session entries are discarded beginning with the oldest first.

  - If the value 0 is specified, the information is not saved.
  - If a value of 1 or greater is specified, that number of session information units is saved.
  - If a negative value is specified, there is no automatic discarding of information.

  If omitted, the value 256 is used.

- loglevel* (log level)

  The log level is specified for all log categories by log level and for each category individually by loglevel_*.

  When the log level for each category has not been specified, the log level for all categories is applied.

  When omitted, the value of the NG_LOG_LEVEL environment variable is used.
  If the NG_LOG_LEVEL environment variable is not set, 2 (Error) is used as the default value of loglevel.

- log_filePath (log file name)

The name of the file to which the log is output is specified in the log file name.

The file name may include a path that includes a directory (e.g., "/home/myHome/var/logFile").

The file and directory name can include the following specifiers.

- "%t"

  "%t" is replaced with the date as year, month and day, and the time in hours, minutes, seconds and milliseconds ("yyyymmdd-hhmmss-MMM") (e.g., "/ home/myHome/var/logDir%t/logFile" is replaced by "/home/myHome/var/logDir20030101-154801-123/logFile").

- "%h"

  "%h" is replaced with the Ninf-G Client hostname.

- "%p"

  "%p" is replaced with the process id of the Ninf-G Client.

When omitted, the log is output to standard error. If the log file name is omitted, the log_suffix, log_nFiles, and log_maxFileSize are ignored.

- log_suffix (log file suffix)

  When a log file is specified, this specifies the suffix used when the log file is created.

  If a suffix is specified, the generated file name will be from "filename[000].suffix" to "filename[nnn].suffix". If omitted, the generated file name will be from "filename.[000]" to "filename.[nnn]". The number of files minus 1 is "nnn."

  The number of digits in "nnn" is the same as the number of digits in number of files minus 1. For example, if the number of files is set to 100, then the number will range from "00" to "99."

- log_nFiles (the number of files for log output)

  This is the number of files created for log output.

  0 indicates that an unlimited number of files can be output. A negative value results in an error.

  If omitted, the value 1 is used.

- log_maxFileSize (maximum number of bytes for the log file)

  This is the maximum number of bytes for the log file.

  If omitted, the value will be unlimited if the number of files is one, or 1Mbyte if the number of files is two or more.

- log_overwriteDirectory (over-write permission for the directory)

  This establishes overwrite permission for the directory. If the specified directory exists, this specifies whether the creation of log files in that directory is enabled or disabled. Operation in the case that the directory exists is shown below.

  - true: No error results, even if the directory specified by log_filePath exists. If files exist in the directory, those file may be overwritten.
  - false: An error results if the directory specified by log_filePath exists.

- tmp_dir (directory for temporal files)

  The directory in which temporal files are placed.

When omitted, TMPDIR environment variable is used if it is defined. Otherwise, "/tmp" is used.

- refresh_credential (Refreshing the proxy credential interval)

  This specifies the interval for refreshing the proxy credential.

  If the value 0 is specified, Ninf-G Client will not refresh the proxy credential. If a value of 1 or greater is specified, Ninf-G Client will refresh the proxy credential and send it to Job Manager. If a negative value is specified, an error results.

  If omitted, the value 0 is used.

  Note: refreshing the proxy credential on a client program feature is supported only in the pthread version. It's OK to build a Ninf-G Executable with both a pthread version and a non-thread version of Globus Toolkit flavor.

  Note: Ninf-G Java Client does not support this feature.

- invoke_server_log (Invoke Server log filename)

  This specifies the Invoke Server log file name. If this attribute is specified, the suffix for Invoke Server name and Invoke Server number is added to the log file name, and then output.

  If omitted, no values are used.

  Note: This attribute appeared in Ninf-G Version 4.0.0.

- fortran_compatible

  This specifies a mode of argument passing.

  By default, a scalar argument for remote functions and remote methods is passed by an immediate value. If this attribute is set to true, a scalar argument is passed by a pointer to the value. Otherwise, a scalar argument is passed as an immediate value.

  If omitted, the default value 'false' is used.

  Note: This attribute has been added in Ninf-G Version 4.1.0.

  Note: Ninf-G Java Client does not support this feature.

- handling_signals

  This attribute specifies signals which will be caught by Ninf-G Client.

  When the Ninf-G Client catches the signal, Ninf-G cleans up all temporary files, cancels all jobs, and exits. This clean up process is performed only for signals which are specified in this attribute.

  The signals are specified by either signal name or signal number. Multiple signals can be specified by space-delimited enumeration. The value "none" can be specified if no signals need to be caught.

  If ommitted, SIGINT, SIGTERM and SIGHUP will be caught by Ninf-G Client.

  Note: This attribute is available for Ninf-G Version 4.2.0 or later.

  Note: Ninf-G Java Client does not support this feature.

- listen_port

  This specifies the client port number for listening requests for unencrypted connections. If the 0 value is specified, an arbitrary port number is used.

If omitted, the default value '0' is used.

Note: This attribute has been added in Ninf-G Version 4.2.0.

- listen_port_authonly

  This specifies the client port number for listening requests for connections by authentication only. If the 0 value is specified, an arbitrary port number is used.

  If omitted, the default value '0' is used.

  Note: This attribute has been added in Ninf-G Version 4.2.0.

- listen_port_GSI

  This specifies the client port number for listening requests for connections encrypted by GSI. If the 0 value is specified, an arbitrary port number is used.

  If omitted, the default value '0' is used.

  Note: This attribute has been added in Ninf-G Version 4.2.0.

- listen_port_SSL

  This specifies the client port number for listening requests for connections encrypted by SSL. If the 0 value is specified, an arbitrary port number is used.

  If omitted, the default value '0' is used.

  Note: This attribute has been added in Ninf-G Version 4.2.0.

- tcp_nodelay (TCP_NODELAY socket option)

  This specifies whether or not to set TCP_NODELAY for both ends of connections between a Ninf-G Client and Ninf-G Executables.

  If omitted, "false" is used.

  Note: This option is available for Ninf-G Version 4.2.3 or later.

  Note: The following is a report from users.

  If the size of transferred arguments or results is less than 1.5KB, performance of data transfer is improved by setting tcp_nodelay to true.

4.3.6 LOCAL_LDIF section

The LOCAL_LDIF section allows multiple definitions. The LOCAL_LDIF section may be omitted.

An example of a LOCAL_LDIF section description is shown below.

```
<LOCAL_LDIF>
filename    file name
filename    file name
</LOCAL_LDIF>
```

The attributes and attribute values of the LOCAL_LDIF section are shown below.

Attribute	Attribute value	Default value	Multiple	Explanation
filename	File name	None	Yes	Local LDIF filename

- filename (local LDIF file)

  This specifies the local LDIF file that contains the Ninf-G Executable information. One file name cannot have multiple descriptions. The local LDIF file is generated by the ng_gen command.

4.3.7 FUNCTION_INFO section

The FUNCTION_INFO section allows multiple definitions. The FUNCTION_INFO section may be omitted.

An example of a FUNCTION_INFO section description is shown below.

```
<FUNCTION_INFO>
hostname      Host name
funcname      Function name

path          Path
staging       [true/false]
backend       Backend
session_timeout Seconds
</FUNCTION_INFO>
```

The attributes and attribute values of the FUNCTION_INFO section are shown below.

Attribute	Attribute value	Default value	Multiple	Explanation
hostname	Host name	None	No	Server machine host name
funcname	Function name	None	No	The function name of the remote function
path	Path	None	No	The path to the Ninf-G Executable
staging	[true/false]	false	No	Staging enabled or disabled
backend	Backend	normal	No	Backend software by which Ninf-G Executable is launched
session_timeout	Seconds	0	No	RPC execution timeout

- hostname (host name of the server machine)

  This specifies the host name of the server machine.

  It cannot be omitted.

- funcname (function name of the remote function)

  This specifies the function name of the remote function.

  It cannot be omitted.

- path (path to Ninf-G Executable)

  This specifies the path to the Ninf-G Executable. If staging is set to true, the path on the client machine is specified. If staging is set to false, the path on the server machine is specified.

  It cannot be omitted.

- staging

  This specifies whether or not staging (*) is to be executed. If 'true' is specified, staging is executed.

  If omitted, the value is taken to be 'false.'

(*) A function for starting up the Ninf-G Executable located on the client machine after transfer to the server machine.

Note: Invoke Server GT4py requires some steps in advance. Details are described in 4.4.1.5 Using staging function on Invoke Server GT4py.

- backend

  This specifies backend software by which the Ninf-G Executable is launched. Backend should be either normal, mpi or blacs. If the backend is normal, Ninf-G Executable is launched directly by GRAM. If the backend is mpi, GRAM will use the mpirun command to launch the Ninf-G Executable as an MPI processes. blacs is used when Ninf-G Executable should be launched by blacs.

  Backend should be specified if neither MDS nor local LDIF is used for execution, and users intend to use mpi or blacs for launching the Ninf-G Executable.

  If omitted, the value is taken to be 'normal.'

- session_timeout

  This specifies the RPC execution timeout value. If the RPC execution time exceeds the timeout value, then the outstanding RPC will be terminated and returned as a timeout error. The handle which was associated with the RPC becomes inoperative and will not be able to be used for any RPCs.

  Measurement of the execution time of an RPC is started when a session invocation API such as grpc_call() is called. The execution time of an RPC involves not only the time for computation of the remote library but also any other unexpected time. For example, the timeout error may occur when the job will not be invoked due to an unknown reason.

  The session_timeout attribute can be used for avoiding unexpected freezes of the Ninf-G Client caused by rare-case accidents on Ninf-G Executables.

  If 0 is specified, then the timeout feature is disabled. The default value of session_timeout is 0.

  Note: The session_timeout feature is supported only for pthreads flavors.

4.3.8 MDS_SERVER section

The MDS_SERVER section allows multiple definitions. The MDS_SERVER section can be omitted.

When an MDS request for information is made, the request is issued to the MDS server specified by the MDS_SERVER section defined in the configuration file. The search is executed repeatedly in order, beginning with the first definition, until the information has been found or the last MDS_SERVER defined in the section has been searched for information.

An example of MDS_SERVER section description is shown below.

```
<MDS_SERVER>
hostname            Host name
tag                 Tag name
type                Type
port                Port number
protocol            protocol name
path                service path
subject             "Subject"
vo_name             VONAME
client_timeout      Seconds
server_timeout      Seconds
</MDS_SERVER>
```

The attributes and attribute values of the MDS_SERVER section are shown below.

Attribute	Attribute value	Default value	Multiple	Explanation
hostname	Host name	None	No	MDS server host name
tag	Tag name	None	No	MDS Server tag name
type	Type	MDS2	No	MDS type
port	Port number	2135	No	MDS server port number
protocol	protocol name	https	No	MDS protocol
path	service path	default service path	No	MDS path
subject	Subject	None	No	MDS subject
vo name	VONAME	Local	No	GIIS vo name
client timeout	Seconds	0	No	The client time-out time
server timeout	Seconds	0	No	The server time-out time

- hostname (the host name of the MDS server)

  This specifies the host name of the MDS server.

  The MDS server host name cannot be omitted.

- tag (MDS Server tag name)

  This specifies the tag name of the MDS Server setting. This tag name is used to specify in a <SERVER> section.

  "tag" for <MDS_SERVER> section has been introduced to allow to define multiple <MDS_SERVER> sections for the same MDS server. Any tag name in a configuration file must be unique.

  If omitted, no values are used.

  Note: This attribute appeared in Ninf-G Version 4.0.0.

- type (MDS type)

  This specifies the type of MDS.

  MDS has 2 types.

  - MDS2 (Pre-WS MDS)
  - MDS4 (Globus Toolkit Information Services)

  If omitted, MDS2 is used.

  Note: This attribute appeared in Ninf-G Version 4.0.0.

- port (the port number of the MDS server)

  This specifies the port number of the MDS server.

  If omitted, the default port (2135 on MDS2, 8443 on MDS4) is used.

- protocol (MDS protocol)

  This specifies the protocol part of the MDS4 URL. (http or https)

  If omitted, https is used.

  Note: This attribute appeared in Ninf-G Version 4.0.0.

- path (MDS path)

  This specifies the path part of the MDS4 URL.

  If omitted, the default service URL path (shown below) is used. /wsrf/services/org/apgrid/ninf/ng4 /grpcinfo/GrpcInfoIndexService

  Note: This attribute appeared in Ninf-G Version 4.0.0.

- subject (MDS subject)

  This specifies the subject used for authentication of MDS4 access to Web Service containers. This is useful for Web Service containers invoked by unprivileged users.

  In order to include space characters, use double quote characters ("). Example: "/C=JP/O=EXAMPLE /OU=GRID/CN=Example of Subject".

  If omitted, this value is not used.

- vo_name (the GIIS vo name)

  This specifies the vo name of GIIS.

  If omitted, "local" is used.

- client_timeout (client time-out value)

  The client time-out value specifies the time-out time for connection between client and server.

  If the value 0 is specified, there is no time-out in waiting for a response.

  If omitted, the default value 0 is used.

- server_timeout (the server time-out value)

  The server time-out value specifies the time-out time for connection between servers.

  If the value 0 is specified, there is no time-out in waiting for a response.

  If omitted, the default value 0 is used.

4.3.9 INVOKE_SERVER section

The INVOKE_SERVER section allows multiple definitions. The INVOKE_SERVER section can be omitted.

Note: This section appeared in Ninf-G Version 4.0.0.

Note: The Invoke Server feature is supported only for pthreads flavors.

An example of an INVOKE_SERVER section description is shown below.

```
<INVOKE_SERVER>
type           type name
path           path name
max_jobs       Number of jobs
log_filePath   path of logfile
status_polling Seconds
option         "option string"
</INVOKE_SERVER>
```

The attributes and attribute values of the INVOKE_SERVER section are shown below.

Attribute	Attribute value	Default value	Multiple	Explanation
type	type name	None	No	Invoke Server type
path	path name	$NG_DIR/bin/ ng_invoke_server.[type]	No	Invoke Server executable file path
max_jobs	Number of jobs	0	No	max jobs for one Invoke Server
log_filePath	path	None	No	log file path
status_polling	Seconds	0	No	Polling interval
option	"option string"	None	Yes	option

- type (Invoke Server type)

  This specifies the type of Invoke Server.

  The default Invoke Server executable file path is $NG_DIR/bin/ng_invoke_server.[type]

  The following four types are available.

  - GT4py : Invoke Server for GT4 (WS GRAM) implemented in Python.
  - SSH : Invoke Server for SSH implemented in C.
  - Condor : Invoke Server for Condor implemented in Java.
  - NAREGISS : Invoke Server for NAREGI Super Scheduler implemented in Java.

  The following three types are available, but unsupported.

  - GT2c : Invoke Server for GT2 (Pre-WS GRAM) implemented in C.
  - GT4java : Invoke Server for GT4 (WS GRAM) implemented in Java.
  - UNICORE : Invoke Server for UNICORE implemented in Java.

  Each Invoke Servers have respective usages and requisites. See 4.4 Invoke Server setup for detail.

  This cannot be omitted.

  Note: The Invoke Server feature is supported only for pthreads flavors.

- path (Invoke Server executable file path)

  This specifies the executable file of Invoke Server.

  The default executable file path is "$NG_DIR/bin/ng_invoke_server.[type]".

  If omitted, the default executable file path are used.

- max_jobs (max jobs for one Invoke Server)

  This specifies the maximum number of jobs for one Invoke Server. If the number of jobs handled reaches the value of max jobs, the next Invoke Server is launched and subsequent jobs are handled by the next Invoke Server.

  If 0 was specified, all job are handled by one Invoke Server.

  If omitted, 0 is used.

- log_filePath (log file path)

  This specifies the log file name for Invoke Server. Nothing are added as suffix of file name.

  If omitted, <CLIENT> section invoke_server_log setting is used.

- status_polling (Polling interval)

This specifies the status polling interval of the Invoke Server. If the Invoke Server implementation uses polling in getting job status, this polling interval is used.

If omitted, 0 is used.

- option (Invoke Server option)

  This specifies the options to pass to Invoke Server, when a function handle is created. Each Invoke Server implementation can define this option for any reason.

  If the value is enclosed by double-quote characters ("), the value can include the space character.

  This attribute can specify multiple times. If omitted, the Invoke Server option is not used.

4.3.10 SERVER section

The SERVER section allows multiple definitions.

When the SERVER section contains multiple definitions, the following API checks to see if remote function information is registered in the first-defined SERVER. If it is, that server is used. If it is not, a check is made for registered remote function information in the second SERVER. This is repeated until remote function information is found.

- grpc_function_handle_default()
- grpc_function_handle_array_default_np()
- grpc_object_handle_default_np()
- grpc_object_handle_array_default_np()

An example of a SERVER section description is shown below.

```
<SERVER>
hostname                        Host name
hostname                        Host name  host name  host name ...

tag                             Tag name
port                            Port number
mds_hostname                    Host name
mds_tag                         Tag name
invoke_server                   Type
invoke_server_option            "option string"
mpi_runNoOfCPUs                 [function name=]number of CPUs
gass_scheme                     [http/https]
crypt                           [false/authonly/SSL/GSI]
protocol                        [XML/binary]
force_xdr                       [true/false]
jobmanager                      JOBMANAGER
subject                         "Subject"
client_hostname                 Host name

job_startTimeout                Seconds
job_stopTimeout                 Seconds
job_maxTime                     Minutes
job_maxWallTime                 Minutes
job_maxCpuTime                  Minutes
job_queue                       Queue name
job_project                     Project name
job_hostCount                   Number of nodes
job_minMemory                   Size
job_maxMemory                   Size
job_rslExtensions               "extension string"

heartbeat                       Seconds
heartbeat_timeoutCount          Times
heartbeat_timeoutCountOnTransfer Times
redirect_outerr                 [true/false]
tcp_connect_retryCount          Counts
tcp_connect_retryBaseInterval   Seconds
tcp_connect_retryIncreaseRatio  Ratio
```

```
tcp_connect_retryRandom        [true/false]
argument_transfer              [wait/nowait/copy]
compress                       [raw/zlib]
compress_threshold             Number of bytes
argument_blockSize             Number of bytes
workDirectory                  Directory name
coreDumpSize                   Size

commLog_enable                 [true/false]
commLog_filePath               File name
commLog_suffix                 Suffix
commLog_nFiles                 Number of files
commLog_maxFileSize            Number of bytes
commLog_overwriteDirectory     [true/false]

debug                          [true/false]
debug_display                  DISPLAY
debug_terminal                 Command path name
debug_debugger                 Command path name
debug_busyLoop                 [true/false]

environment                    Variable name
environment                    Variable name = value
</SERVER>
```

The attributes and attribute values of the SERVER section are shown below.

Attribute	Attribute value	Default value	Multiple	Explanation
hostname	Host name	None	Yes	Server machine host name
tag	Tag name	None	No	Server tag name
port	Port number	2119	No	The server port number on which the Globus gatekeeper is listening
mds_hostname	Host name	None	No	MDS server host name
mds_tag	Tag name	None	No	MDS tag name
invoke_server	type	None	No	Invoke Server type
invoke_server_option	option string	None	Yes	Invoke Server option
mpi_runNoOfCPUs	Function name, Number of CPUs	None	Yes	The number of CPUs used by MPI function
gass_scheme	[http/https]	http	No	GASS server scheme
crypt	[false/authonly /SSL/GSI]	false	No	Method of authentication and encryption for communication paths
protocol	[XML/binary]	XML	No	Specifies the protocol.
force_xdr	[true/false]	false	No	Makes XDR compulsory.
jobmanager	JOBMANAGER	None	No	The job manager used on the server machine
subject	Subject	None	No	Subject of resource manager contact
client_hostname	Host name	hostname of CLIENT section	No	Client machine host name
job_startTimeout	Seconds	0	No	The time-out at job startup
job_stopTimeout	Seconds	-1	No	The time-out for when the job stops

job_maxTime	Minutes	None	No	The maximum job execution time
job_maxWallTime	Minutes	None	No	The maximum job execution wall clock time
job_maxCpuTime	Minutes	None	No	The maximum job execution cpu time
job_queue	queue name	None	No	A remote queue name
job_project	project name	None	No	A remote project name
job_hostCount	Number of nodes	None	No	Number of nodes (for SMP clusters)
job_minMemory	Size	None	No	Minimum amount of memory, in Megabytes
job_maxMemory	Size	None	No	Maximum amount of memory, in Megabytes
job_rslExtensions	extension string	None	No	RSL extensions
heartbeat	Seconds	60	No	The heart-beat interval
heartbeat_timeoutCount	Times	5	No	The heart-beat time-out times
heartbeat_timeoutCountOnTransfer	Times	5	No	The heart-beat time-out times on transfer
redirect_outerr	[true/false]	true	No	Ninf-G Executable output redirect
tcp_connect_retryCount	Counts	4	No	The maximum number of retries for a TCP connection
tcp_connect_retryBaseInterval	Seconds	1	No	The base interval time for the first retry
tcp_connect_retryIncreaseRatio	Ratio	2.0	No	The increase ratio for calculating the maximum interval time between retries
tcp_connect_retryRandom	[true/false]	true	No	A flag that specifies whether the random value is used or not for the interval time
argument_transfer	[wait/nowait /copy]	wait	No	Returns the called function for an asynchronous function call Timing (Wait or do not wait for completion of argument transfer.)
compress	[raw/zlib]	raw	No	Compression method
compress_threshold	Number of bytes	64KBytes	No	Threshold for performing compression
argument_blockSize	Number of bytes	16KBytes	No	The block size of transferred arguments
workDirectory	Directory name	The path to the Ninf-G Executable	No	The working directory for Ninf-G Executable
coreDumpSize	Size	Undefined	No	Core dump size for Ninf-G Executable

commLog_enable	[true/false]	false	No	Whether the communication log output is enabled or disabled
commLog_filePath	File name	stderr	No	Communication log file name
commLog_suffix	Suffix	Sequence number	No	The communication log file suffix
commLog_nFiles	Number of files	1	No	The number of files for communication log output
commLog_maxFileSize	Number of bytes	1M/unlimited	No	Maximum number of bytes for the communication log file
commLog_overwriteDirectory	[true/false]	false	No	Overwrite permission for the communication log directory
debug	[true/false]	false	No	Whether the debugging function is enabled or not
debug_display	DISPLAY	Environment variable	No	Debugging display
debug_terminal	Command path name	Environment variable	No	Path to the debugging terminal emulator
debug_debugger	Command path name	Environment variable	No	Debugger path
debug_busyLoop	[true/false]	false	No	Wait for attach from debugger or not
environment	Character string	None	Yes	Environment variable

- hostname (the host name of the server machine)

  This specifies the host name of the server machine.
  Multiple hostname attributes can be defined.
  It is possible for multiple host names to be defined on one line.
  This value cannot be omitted.

- tag (Server tag name)

  This specifies the tag name of a <SERVER> section.

  "tag" for <SERVER> section has been introduced to allow to define multiple <SERVER> sections for the same server. APIs which create function handles or object handles accept tag name as well as hostname as the host name of the server. Any tag name in a configuration file must be unique.

  The tag name can include neither '/' nor ':' character since those characters are reserved characters of Resource Manager Contact.

  If omitted, no values are used.

  Note: This attribute is available for Ninf-G Version 4.0.0 or later.

- port

  This specifies the port number on which the Globus gatekeeper is listening.

  If omitted, the default '2119' value is used.

- mds_hostname (the host name of the MDS server)

This is the MDS queried first when querying for information concerning the MPI of the server machine and remote function information. This specifies the host name of the server.

If the mds_hostname attribute is specified, the mds_tag attribute cannot be specified.

If omitted, no values are used.

- mds_tag (MDS tag name)

  This specifies the MDS tag name for the MDS server setting. If the mds_tag attribute is specified, the mds_hostname attribute cannot be specified.

  If omitted, no values are used.

  Note: This attribute appeared in Ninf-G Version 4.0.0.

- invoke_server (Invoke Server type)

  This specifies the Invoke Server type to use for the server.

  The attribute arguments are described in the <INVOKE_SERVER> section type attribute.

  If omitted, the Invoke Server is not used.

  Note: This attribute appeared in Ninf-G Version 4.0.0.

  Note: The Invoke Server feature is supported only for pthreads flavors.

- invoke_server_option (Invoke Server option)

  This specifies the options to pass to Invoke Server, when a function handle is created. Each Invoke Server implementation can define this option for any reason.

  If the value is enclosed by double-quote characters ("), the value can include the space character.

  This attribute can be specified multiple times. If omitted, the Invoke Server option is not used.

  Note: This attribute appeared in Ninf-G Version 4.0.0.

- mpi_runNoOfCPUs (number of MPI CPUs)

  This specifies the number of CPUs to be used when MPI is used on a server machine.

  The number of CPUs for executing particular functions can be specified with the format "function name = number of CPUs."

  If the function name is omitted, the default value for the number of CPUs for MPI on that server machine is used.

- gass_scheme (GASS server scheme)

  This specifies the scheme for the GASS server.

  If omitted, http is used.

- crypt (method of authentication and encryption for communication paths)

  This specifies the method of authentication and encryption for communication paths. Choices are no authentication and no encryption (false), authentication only (authonly), authentication and encryption using SSL (SSL), and authentication and encryption using GSI (GSI).

  If omitted, the default value 'false' is used.

Note: Encryption of transferred data is implemented using GSI, hence proxy certificate must be delegated to Ninf-G Executable. Therefore, encryption is basically available only for Pre-WS GRAM and WS GRAM.

Note: 'authonly' is available in Ninf-G Version 4.2.0 or later.

- protocol (protocol specification)

  This specifies the protocol to be used between a Ninf-G Client and Ninf-G Executable. Either XML or binary can be specified.

  If omitted, XML is used.

- force_xdr (whether or not to force XDR)

  This specifies whether or not to force the use of XDR in the protocol between a Ninf-G Client and Ninf-G Executable.

  If XML is used as the protocol, this setting has no effect. The main purpose of using this is for measurement of processing speed when XDR is used.

  If omitted, the default value 'false' is used.

- jobmanager (the job manager to be used on the server machine)

  This specifies the job manager to be used on the server machine. Any of jobmanager-fork, jobmanager-pbs, jobmanager-gdr, or jobmanager-lsf can be specified, depending on the server machine settings.

  If omitted, the default job manager on the server machine is used.

- subject (subject part of GRAM resource manager contact)

  This specifies the subject part of the Globus Toolkit GRAM resource manager contact. The subject is usually used for the Globus personal gatekeeper.

  If the value is enclosed by double-quote characters ("), the value can include the space character, as in "/C=JP/O=EXAMPLE/OU=GRID/CN=Example of Subject".

  If omitted, no value is used.

- client_hostname (the host name of the client machine)

  This specifies the host name of the client machine.

  The Ninf-G Executable on the server will connect back to the client machine which is specified by this attribute.
  The attribute enables each server to use different names of the client machine according to the network configuration of the client and the servers.

  If omitted, hostname of CLIENT section is used.

  Note: This attribute is not available if you use PreWS GRAM and enable redirect_outerr or executable staging.

- job_startTimeout (the job startup time-out)

  This specifies the time-out time for job startup.

  When grpc_call(), grpc_invoke_np() or another such RPC is executed, if the job has not started after this time has passed since the job start request was issued, a time-out occurs; each API ends and returns an error.

If the 0 value is specified, there is no time-out and the process waits until the job starts. If a value of 1 or greater is specified, the process waits that amount of time for the job to start. If a negative value is specified, an error results.

If omitted, the 0 value is used.

- job_stopTimeout (the job stop time-out time)

  When grpc_function_handle_destruct(), grpc_object_handle_destruct() or other such job stop request is issued by the API, if the job has not stopped after this time elapses, a time-out occurs; each API ends and returns an error.

  If a negative value is specified, there is no time-out and the process waits until the job stops. If the 0 value is specified, the process doesn't wait for the job to stop. If a value of 1 or greater is specified, the process waits that amount of time for the job to stop.

  If omitted, the -1 value is used.

  Note: This attribute changed in Ninf-G Version 2.4.0.
  In Ninf-G Version 2.3.0 or former:
  If the 0 value is specified, there is no time-out and the process waits until the job stops. If a negative value is specified, an error results.

- job_maxTime (the maximum job execution time)

  This specifies the maximum job execution time. The value specified is used to pass the Globus GRAM RSL attribute "maxTime." The units are in minutes.

  If omitted, no values are used.

- job_maxWallTime (the maximum job execution wall clock time)

  This specifies the maximum job execution wall clock time. The value specified is used to pass the Globus GRAM RSL attribute "maxWallTime." The units are in minutes.

  If omitted, no values are used.

- job_maxCpuTime (the maximum job execution cpu time)

  This specifies the maximum job execution cpu time. The value specified is used to pass the Globus GRAM RSL attribute "maxCpuTime." The units are in minutes.

  If omitted, no values are used.

- job_queue (queue name)

  Target the GRAM job to a queue (class) name as defined by the scheduler at the defined (remote) resource.

  If omitted, no values are used.

- job_project (project name)

  Target the GRAM job to be allocated to a project account as defined by the scheduler at the defined (remote) resource.

  If omitted, no values are used.

- job_hostCount (number of nodes)

  Defines the number of nodes (hosts) to distribute the Ninf-G Executable processes created by handle array init API across. This attribute only applies to clusters of SMP computers.

If omitted, no values are used.

Note: There is a bug in jobmanager-pbs, so jobmanager-pbs doesn't work with this attribute variable.

- job_minMemory (minimum amount of memory)

  Specify the minimum amount of memory required for a Ninf-G Executable process. Units are in Megabytes.

  If omitted, no values are used.

- job_maxMemory (maximum amount of memory)

  Specify the maximum amount of memory required for a Ninf-G Executable process. Units are in Megabytes.

  If omitted, no values are used.

- job_rslExtensions (RSL extensions)

  This specifies the WS GRAM RSL extensions. This attribute is available for Invoke Server GT4py and for PreWS GRAM.

  WS GRAM RSL extensions is currently used only to specify client-specific data which the client wishes to associate with the job it is controlling.

  WS GRAM RSL extensions can be processed by user defined WS GRAM jobmanager scripts. For Globus Toolkit 4.0.1, calling $description->extensions() subroutine in the file $GLOBUS_LOCATION/lib/perl/Globus/GRAM/JobManager/fork.pm implements accessibility to the given RSL extensions. (See Globus Toolkit WS GRAM Users Guide for details.)

  In addition, this attribute is also used to specify user defined PreWS GRAM attributes. Attribute values will just be added to the end of the RSL.

  If the attribute value is enclosed by double-quote characters ("), the value can include the space and other characters.

  In the string enclosed by double-quote characters, some characters are considered as escape characters.

  - A backslash double-quote (¥") denotes a double-quote character (") in the value.
  - A backslash backslash (¥¥) denotes backslash character (¥) in the value.
  - A backslash return denotes that, the attribute value continues to the next line.
  - A backslash followed by the other characters causes an error.

  Here is an example valid usage.

```
job_rslExtensions ” ¥
    <myAttribute> ¥
    ¥”test¥¥ value¥” ¥
    </myAttribute>”
```

  This attribute can be specified multiple times. If omitted, job_rslExtensions is not used.

  Note: This attribute is available for Ninf-G Version 4.0.0 or later.

- heartbeat (the heart-beat interval)

  This specifies the interval for sending the heart-beat from Ninf-G Executable to Ninf-G Client.

If the value 0 is specified, the heart-beat is not sent. If a value of 1 or greater is specified, the heartbeat is sent at that interval. If a negative value is specified, an error results.

If omitted, the value 60 is used.

Note: The heartbeat checking on a client program feature is supported only in the pthread version. The heartbeat sending on a Ninf-G Executable feature is supported by both the pthread version and the non-thread version of Globus Toolkit flavor.

Note: If you are debugging a Ninf-G Executable or client, We suggest that you disable the heartbeat feature. This is to suppress periodic heartbeat overhead and unexpected heartbeat timeouts.

- heartbeat_timeoutCount (the heart-beat time-out time)

  This specifies the number of times until a time-out occurs when the heart-beat is not being sent.

  When the heartbeat has not been sent for a time equal to the heart-beat interval times the heart-beat time-out value, the Ninf-G Client takes it as meaning that the Ninf-G Executable is also not operating.

  If omitted, the value 5 is used.

- heartbeat_timeoutCountOnTransfer (the heart-beat time-out time on transfer)

  This specifies the number of continuous lost heartbeat messages for detecting heartbeat-timeout errors during data transfer between Ninf-G Client and Ninf-G Executable.

  In the current Ninf-G implementation, heartbeat message is not sent while transferring data (input/output arguments of RPC). Therefore, if the data is large and it takes long time for the data transfer, heartbeat is not sent for long time hence heartbeat timeout error may occur. In order to avoid this problem, this attribute is provided to set heartbeat timeout count specific for data transfer. Set 0 to ignore heartbeat timeout during the data transfer, or set large value which is large enough to avoid unexpected heartbeat timeout error.

  If ommitted, the same value with heartbeat_timeoutCount attribute is used.

  Note: This attribute is available for Ninf-G Version 4.2.0 or later.

- redirect_outerr (redirection of the Ninf-G Executable output)

  This specifies redirection of the standard error or standard output of a Ninf-G Executable to a Ninf-G Client.

  If omitted, the value 'true' is used.

  Note: If the save_stdout or the save_stderr attribute on the server side configuration file is set, stdout or stderr is not delivered to the Ninf-G Client regardless of the value of redirect_outerr.

- tcp_connect_retryCount (Retry count for TCP connect)

  This specifies the maximum number of retries for establishing a TCP connection. This attribute is used for the following cases.

  - A connection for a connect back from a Ninf-G Executable to a client.
  - A connection for GASS file transfers with FILENAME type arguments.

  The default value of this attribute is 4.

  Note: Ninf-G 2.3.0 and prior versions do not support this attribute. In order to disable this attribute, set tcp_connect_retryCount to 0 if the version of Ninf-G on the server is 2.3.0 or prior.

- tcp_connect_retryBaseInterval (Retry base interval for TCP connect)

  This specifies the base interval time for the first retry. The value is in seconds and must be a non-negative integer. This value is used as the maximum interval time for the first retry.

  The default value of this attribute is 1.

- tcp_connect_retryIncreaseRatio (Retry increase ratio for TCP connect)

  This specifies the increase ratio which is used to calculate the maximum interval time between retries. The maximum interval time is calculated by multiplying this value and the maximum interval time for the last retry. For the first retry, the value of tcp_connect_retryBaseInterval is used as the maximum interval time.

  The value must be greater than 1.0 and the default value of this attribute is 2.0.

- tcp_connect_retryRandom (Random for TCP connect)

  This specifies a flag that specifies whether a random value is used or not for the interval time. If the value is true, the interval time between retries is set randomly between 0.0 seconds to the maximum interval time. If the value is false, the maximum interval time is used as the interval time.

  The default value of this attribute is true.

- argument_transfer (the timing for the return of the calling function for an asynchronous call)

  When an asynchronous call function is used, this specifies the timing for that function's return.

  The values that can be specified are 'wait' (wait until argument transfer is completed), 'nowait' (do not wait until argument transfer is completed), and 'copy' (without waiting for the completion of argument transfer, the values of the arguments passed to the asynchronous function are copied on the client side, and the argument transfer is done in the background).

  If omitted, 'wait' is used.

- compress (compression method)

  This specifies the method for compressing the argument information. Either 'raw' or 'zlib' can be specified.

  If omitted, 'raw' is used.

- compress_threshold (the threshold value for performing compression)

  This specifies the threshold value when compression is performed. If the argument information size equals or exceeds the specified value, the information is compressed.

  If omitted, the value of 64 kilobytes is used.

- argument_blockSize (The argument block size)

  Arguments and results are divided into a specified block size when they are transferred between a Ninf-G Client and a Ninf-G Executable.

  The value of this attribute affects the performance of data transfer and an appropriate value should be specified according to the size of the transferred data and network performance.

  If 0 is specified, arguments and results will not be divided. If a positive integer is specified, they are divided into blocks with the specified value. An error occurs if a negative value is specified.

  If omitted, the default value 16Kbytes is used.

- workDirectory (the working directory for the Ninf-G Executable)

  This specifies the working directory for the Ninf-G Executable.

  If omitted, no changing for the working directory is made when the staging function is used, in any other case, the Ninf-G Executable path is used for the working directory.

- coreDumpSize (core dump size for Ninf-G Executable)

  This specifies the core dump file size for the Ninf-G Executable. The size is in 1024-byte increments.

  If 0 is specified, it means no core dump file is created. If -1 is specified, it means core dump file size is unlimited and infinite.

  If omitted, no setup for core dump file size is performed.

- commLog_enable (whether communication log output is enabled or disabled)

  This specifies whether the communication log output function is enabled or disabled.

  If 'true' is specified, the communication log is output.

  If not specified, the default value is false.

- commLog_filePath (communication log file name)

  The name of the file to which the communication log is output is specified in the log file name.

  The file name may include a path that includes a directory (e.g., "/home/myHome/var/logFile").

  The file and directory name can include the following specifiers.

  - "%t"

    "%t" is replaced with the date as year, month and day, and the time in hours, minutes, seconds and milliseconds ("yyyymmdd-hhmmss-MMM") (e.g., "/ home/myHome/var/logDir%t/logFile" is replaced by "/home/myHome/var/logDir20030101-154801-123/logFile").

  - "%h"

    "%h" is replaced with the Ninf-G Client hostname.

  - "%p"

    "%p" is replaced with the process id of the Ninf-G Client.

  The Ninf-G Executable id number is added to the end of the file name.

  When omitted, the log is output to standard error. If the communication log file name is omitted, commLog_suffix, commLog_nFiles, and commLog_maxFileSize are ignored.

- commLog_suffix (communication log file suffix)

  When the communication log file is specified, this specifies the suffix used when the log file is created.

  If a suffix is specified, the generated file name will be from "filename[000].suffix" to "filename[nnn].suffix". If omitted, the generated file name will be from "filename.[000]" to "filename.[nnn]". The number of files minus 1 is "nnn." The number of digits in "nnn" is the same as the number of digits in the number of files minus 1. For example, if the number of files is set to 100, then the number will range from "00" to "99."

- commLog_nFiles (number of files for communication log output)

This is the number of files created for communication log output.

0 indicates an unlimited number of files can be output. A negative value results in an error.

If omitted, the value 1 is used.

- commLog_maxFileSize (maximum number of bytes for the communication log file)

  This specifies the maximum number of bytes for the communication log file.

  If omitted, the value will be unlimited if the number of files is one, or 1Mbyte if the number of files is two or more.

- commLog_overwriteDirectory (over-write permission for the directory)

  This establishes overwrite permission for the directory. If the specified directory exists, this specifies whether creation of log files in that directory is enabled or disabled. Operation in the case that the directory exists is shown below.

  - true: There is no error even if the directory specified by log_filePath exists. It is possible that files located in that directory will be overwritten.
  - false: If the directory specified by log_filePath exists, an error results.

- debug (debugging function enabled or disabled)

  This specifies whether the debugging function is enabled or disabled.

  If 'true' is specified, the debugger will be started up when the Ninf-G Executable starts up, allowing debugging of the Ninf-G Executable. If 'false' is specified, the Ninf-G Executable starts up without starting the debugger.

  If omitted, the default 'false' value is used.

- debug_display (debugging display)

  This specifies an X11 display for displaying the debugging terminal emulator.

  To use the debugger, start up the terminal emulator on the server machine, and run the debugger on that terminal. This defines the value for the environment variable DISPLAY that is passed to the terminal emulator.

- debug_terminal (the path to the debugging terminal emulator)

  This specifies the path to the terminal emulator command.

  If omitted, the value 'xterm' is used. The Ninf-G Executable searches for terminal emulator command in PATH that is set in the Ninf-G operating environment on the server machine that is used.

- debug_debugger (path to the debugger)

  This specifies the path to the debugger command.

  If omitted, the value 'gdb' is used. The Ninf-G Executable searches for the debugger command in PATH that is set in the Ninf-G operating environment on the server machine that is used.

- debug_busyLoop (wait attach from debugger)

  This specifies whether the Ninf-G Executable perform waiting attach from the debugger or not.

  If 'true' is specified, the Ninf-G Executable waits for attaching from the debugger, just after its invocation.

The user needs to invoke the debugger and attach that Ninf-G Executable. Then the user must change the variable for waiting attach (debugBusyLoop), and continue execution. (When the user uses gdb, try "set var debugBusyLoop=0", "continue".)

If omitted, the default 'false' value is used.

Note: It's very helpful to specify this attribute with "environment NG_LOG_LEVEL=4" in the SERVER section. which displays which process id must be attached.

- environment (environment variable)

  The environment variable specifies the environment variable that is passed to the Ninf-G Executable. It can be written as 'variable name' only or 'variable name = value' style.

  If omitted, the environment variable is not used.

4.3.11 SERVER_DEFAULT section

The SERVER_DEFAULT section does not allow multiple definitions. This section may be omitted.

The SERVER_DEFAULT section defines the default values for attributes which are used when attributes are omitted in the SERVER section.

The description of the SERVER_DEFAULT section is the same as the SERVER section, except that the attribute "hostname" is not described.

The SERVER_DEFAULT section may also be described in the configuration file or other such places. (*)

(*) For example, even if the SERVER_DEFAULT section is written later than the SERVER section, if attributes are omitted in the previously described SERVER section, the attributes defined in the SERVER_DEFAULT section are used.

4.4 Invoke Server setup

Ninf-G4 implements mechanisms for remote process invocation as a separate module called Ninf-G Invoke Server. This architecture enables to support any job submission interfaces by implementing Ninf-G Invoke Server for the interface.

Users must specify the Invoke Server for each server in Ninf-G Client Configuration file except for Pre-WS GRAM. RPC mechanisms for Pre-WS GRAM is embedded in Ninf-G Library and it is not necessary to use Invoke Server for Pre-WS GRAM.

Here is an example of the description of <SERVER> section in the Ninf-G Client Configuration file for specifying WS GRAM as a job submission interface.

```
<SERVER>
hostname your-host
invoke_server GT4py
</SERVER>
```

Invoke Server can be set and configured in the Ninf-G Client Configuration file as described above. The details of the configuration of Invoke Server are described in sections 4.3.9 and 4.3.10.

Each Invoke Server may have its own options. In order to specify such options, the following attributes are provided in the Ninf-G Client Configuration file.

- "invoke_server_option" attribute in <SERVER> section

This attribute is used to specify Invoke Server options for a specific server.

- "option" attribute in <INVOKE_SERVER> section

  This attribute is used to specify Invoke Server options for all servers.

Example:

```
<SERVER>
hostname your-host
invoke_server GT4py
invoke_server_option "delegate_full_proxy true"
</SERVER>
```

Some attributes in <SERVER> section are interpreted by each Invoke Server. For example, Invoke Server GT4py interprets "port" attribute as the port number of WS GRAM and Invoke Server SSH interprets "port" attribute as the port number of sshd.

Note: The Invoke Server feature is supported only for pthreads flavors.

4.4.1 Invoke Server GT4py

Invoke Server GT4py invokes Ninf-G Executable via WS GRAM.

1. Prerequisite

   GT4 must be installed on both client and server. `globusrun-ws` command must be available on the client and remote server must be able to accept WS GRAM access.

2. Install

   Invoke Server GT4py is automatically installed through the Ninf-G installation processes described in section 2 of this manual.

3. Extra options

   Invoke Server GT4py accepts the following extra options.

   - delegate_full_proxy

     This attribute specifies the type of delegated proxy certificate. If delegate_full_proxy is set to "true", full proxy certificate is delegated to the server. Otherwise, limited proxy certificate is delegated. This option is provided for enabling cascading RPC since limited proxy certificate does not allow subordinate GRAM accesses.

     If omitted, "false" is used.

     Note: If this option is set to true, extra command (`globus-credential-delegate`) is executed internally that may take 10 to 20 seconds if Globus Toolkit Version 4.0.3 or prior is used.

     Note: This option is available for Ninf-G Version 4.2.0 or later.

   - protocol

     This attribute specifies the protocol to WS GRAM. If WS GRAM is non secure mode (started by `globus-start-container -nosec`), "protocol http" must be set to access the WS GRAM.

     If ommitted, "https" is used.

4. RSL extensions

WS GRAM RSL has <extensions> tag, which enables the user to pass extra information to WS GRAM server.

Invoke Server handles this feature by using job_rslExtensions attribute in <SERVER> section.

5. Using staging function

   Executable staging on WS GRAM server via Invoke Server GT4py requires the following steps in advance.

   1. Invoke GridFTP servers on both server and client hosts.

      Invoke Server GT4py requires GridFTP servers on both remote and local hosts. The GridFTP server should be invoked either directly or via inetd/xinetd daemon. The port for the GridFTP server is not limited to the default port 2811.

   2. Specify the port number for client-side GridFTP server in the client configuration file.

      If the client-side GridFTP server does not use the default port (2811), the port number of the GridFTP server must be specified in client configuration file. The port number can be specified by gsiftp_port option in invoke_server_option attribute in <SERVER> section.

      ```
      example:
      <SERVER>
      invoke_server GT4py
      invoke_server_option "gsiftp_port 12811"
      ...
      </SERVER>
      ```

   3. Specify subject name for authentication.

      Subject names which are used for mutual authentication between WS GRAM container and client-side GridFTP server depends on the owner of those daemons.

      If they are invoked by the system, subject name of the host certificate is used. If they are invoked by a user, subject name of the user certificate is used.

      According to the combination of the owners of the WS GRAM container and the client-side GridFTP server, some attributes need to be specified in the client configuration file.

      - Case 1: Both the WS GRAM container and the client-side GridFTP server are run by the system.

        It is not necessary to specify the subject name.

      - Case 2: The WS GRAM container is run by the system and the client-side GridFTP server is run by a user.

        The subject name of the user must be specified by staging_source_subject attribute in <SERVER> section.

        ```
        <SERVER>
        invoke_server_option "staging_source_subject /Subject/of/User"
        ...
        </SERVER>
        ```

      - Case 3: Both the WS GRAM container and the client-side GridFTP server are run by a user.

The subject name of the user must be specified by subject attribute in <SERVER> section.

```
<SERVER>
subject "/Subject/of/User"
...
</SERVER>
```

- Case 4: The WS GRAM container is run by a user and the client-side GridFTP server is run by the system.

  The subject name of the user must be specified by subject attribute in <SERVER> section. The subject name of the client-side host must be specified by staging_source_subject attribute in <SERVER> section. The subject name of the user must be specified by staging_destination_subject and deletion_subject attributes in <SERVER> section.

```
<SERVER>
subject "/Subject/of/User"
invoke_server_option "staging_source_subject /Subject/of/ClientHost"
invoke_server_option "staging_destination_subject /Subject/of/User"
invoke_server_option "deletion_subject /Subject/of/User"
...
</SERVER>
```

4. Create a scratch directory for the server.

   For each server host, the scratch directory must be created for staging in advance.

   The staging directory is "$HOME/.globus/scratch" ($GLOBUS_SCRATCH_DIR variable in GT4 GRAM RSL).

   Please create the directory as follows:

```
% mkdir ~/.globus/scratch
% chmod 700 ~/.globus/scratch
```

4.4.2 Invoke Server SSH

Invoke Server SSH invokes Ninf-G Executable via SSH.

1. Prerequisite

   User must be able to execute commands on the server using ssh command. In addition, it is recommended to configure user's ssh environments not to require user's input (e.g. password) for executions to avoid repetitious input while Ninf-G application is executed. "`ssh-agent`" and "`ssh-add`" commands are usually used for such purposes.

   The following commands are required by Invoke Server SSH and must be available on the server.

   `/bin/sh`, `/bin/echo`, `/bin/grep`, `/bin/chmod`, `/bin/mkdir`, `/bin/cat`, `/bin/rm`, `/bin/kill`

2. Install

   Invoke Server SSH is automatically installed through the Ninf-G installation processes described in section 2 of this manual.

3. Job submission system

   Like Globus GRAM, Invoke Server SSH is able to launch remote processes via a backend queuing system including SGE and PBS(*1). The backend queuing system is specified by "jobmanager" attribute in <SERVER> section in the Ninf-G Client Configuration file. The value of "jobmanager" attribute can be either "jobmanager-sge" for SGE or "jobmanager-pbs" for PBS.

   Example:

```
<SERVER>
   hostname example.org
   invoke_server SSH
   jobmanager jobmanager-sge
      :
      :
</SERVER>
```

   It should be noted that although the values "jobmanager-sge" and "jobmanager-pbs" are also used for Invoke Servers for Globus GRAM (e.g. GT4py), jobmanager programs used by Invoke Server SSH are implemented by the Ninf-G development team hence they are completely different with the jobmanager programs provided by the Globus Toolkit.

   The jobmanager program assumes that user's home directory is shared between front (master) node and compute nodes.

   Invoke Server SSH uses `qsub`, `qstat`, and `qdel` commands in jobmanager-sge and jobmanager-pbs. Therefore, the path of these commands should be included in PATH environment variable. Otherwise, the path of these commands must be passed by options described below.

Command	Option
qsub	ssh_submitCommand
qstat	ssh_statusCommand
qdel	ssh_deleteCommand

   The detailed description of these options is described in section 4.4.2.4 of this manual.

   (*1) Invoke Server SSH is tested with PBS Pro and Torque.

4. Extra options

   Invoke Server SSH accepts the following extra options.

   - ssh_command

     This option specifies the path of "`ssh`" command. Invoke Server SSH connects to remote host using the command specified by this attribute.

     If omitted, `/usr/bin/ssh is used.`

   - ssh_remoteSh

     This option specifies the path of shell command to invoke shell on remote host. If backend queuing system is used, the specified shell is also used in the script for backend queuing system.

     If omitted, `/bin/sh` is used.

   - ssh_user

This option specifies the user name on remote host. This value is passed to "`ssh`" command as "`-l`" argument.

If omitted, "`-l`" option is omitted.

- ssh_option

  This option specifies the any options which will be passed to "`ssh`" command. Multiple ssh_option options can be specified.

- ssh_remoteTempdir

  This option specifies the directory in which temporary files are created on remote host.

  If omitted, home directory is used.

- ssh_submitCommand

  This option specifies the command for submitting jobs on remote host. This option is available only when backend queuing system is used.

  If omitted, `qsub` is used.

- ssh_statusCommand

  This option specifies the command for querying status of jobs on remote host. This option is available only when backend queuing system is used.

  If omitted, `qstat` is used.

- ssh_deleteCommand

  This option specifies the command for deleting jobs on remote host. This option is available only when backend queuing system is used.

  If omitted, `qdel` is used.

- ssh_MPIcommand

  This option specifies the command for launching a MPI program on remote host. This command is used when Invoke Server SSH invokes MPI jobs.

  If omitted, "`mpirun`" is used.

- ssh_MPIoption

  This option specifies the command line options which will be passed to "`mpirun`" command on remote host. This is used when Invoke Server SSH invokes MPI jobs. Multiple ssh_MPIoption options can be defined.

- ssh_MPInumberOfProcessorsOption

  This option specifies the command line option of `mpirun` command for specifying the number of processors. This option is used when Invoke Server SSH invokes MPI jobs.

  The value of this option must include "`%d`" and Invoke Server SSH replaces it by the actual number of processors.

  If omitted, "`-np %d`" is used.

- ssh_MPImachinefileOption

This option specifies the command line option of "`mpirun`" command for specifying machinefile. This option is used when Invoke Server SSH invokes MPI jobs using backend queuing system.

The value of this option must include "`%s`" and Invoke Server SSH replaces it by the name of the actual machinefile.

If omitted, "`-machinefile %s`" is used.

- ssh_SGEparallelEnvironment

  This option specifies the parallel environment of SGE. It is used when Invoke Server SSH invokes MPI jobs or array jobs using SGE.

  If omitted, `*mpi*` is used.

- ssh_PBSprocessorsPerNode

  This option specifies the number of processors per a node. It is used when Invoke Server SSH invokes MPI jobs or array jobs using PBS.

  If omitted, 1 is used.

- ssh_PBSrsh

  This specifies the RSH command used on remote host when Invoke Server SSH invokes array jobs using PBS.

  If omitted, `/usr/bin/ssh` is used.

4.4.3 Invoke Server Condor

Invoke Server Condor invokes Ninf-G Executable via Condor(*1).

*1 Condor Project: http://www.cs.wisc.edu/condor/

1. Prerequisite

   - Condor 6.6.11 or later (unconfirmed, older than 6.6.11)

     Condor must be installed on both client and server machines.

   - JDK 5.0 or later

2. Install

   Invoke Server Condor is not installed by the default Ninf-G installation and it must additionally installed manually according to the following steps.

   1. Set the NG_DIR environment variable

      ```
      csh.
      % setenv NG_DIR /path/to/ninf-g

      sh.
      $ NG_DIR=/path/to/ninf-g ; export NG_DIR
      ```

   2. Change directory to the directory of Invoke Server Condor.

      ```
      % cd ng-4.x.x  # expanded Ninf-G package
      % cd utility/invoke_server/condor
      ```

3. Run "`make`" command to compile Invoke Server Condor

   ```
   % make
   ```

4. Run "`make install`" command to install Invoke Server Condor

   ```
   % make install
   ```

   This command copies the following files under ${NG_DIR} directory.

   `${NG_DIR}/lib/`

   - `classad.jar` - Log analysis library for Condor Job
   - `condorAPI.jar` - Condor Java API Library
   - `condorIS.jar` - Invoke Server Condor

   `${NG_DIR}/bin/`

   - `ng_invoke_server.Condor` - Startup script for Invoke Server Condor

3. Extra options

   none

4. Information

   Invoke Server Condor automatically creates the Condor job cluster log when it invokes jobs. The name of the log file is "`ninfg-invoke-server-condor-log`".

5. Limitation

   - Invoke Server Condor supports vanilla universe only.
   - MPI job is not supported.

4.4.4 Invoke Server NAREGISS

Invoke Server NAREGISS invokes Ninf-G Executable via NAREGI Super Scheduler.

1. Prerequisite

   NAREGI Middleware V1.1 or later is required. Java 1.5.0 or later.

2. Install

   Invoke Server NAREGISS can be installed as a part of Ninf-G installation steps. Invoke Server NAREGISS is installed if `--with-naregi` is specified as a Ninf-G configure script option.

   Example:

   ```
   % ./configure --with-naregi
   ```

   NOTE: If NAREGI Middleware is not installed in default the directory (/usr/naregi), it is necessary to specify it with configuration option "--with-naregidir".

   Details of Ninf-G configure script are described in 2.4 Configure command options.

3. Note

Invoke Server NAREGISS assumes that the Ninf-G Client is invoked as a job via NAREGI SS, and expects the followings.

- URL of the NAREGI SS server is set as environment variable NAREGI_GRIDSS_URL.
- Renewal Service hostname and port number is set as environment variable NAREGI_RENEWAL_HOSTPORT.
- My Proxy server hostname and port number is set as environment variable NAREGI_MYPROXY_HOSTPORT.
- VOMS compliant proxy credential

4. Extra options

   Invoke Server NAREGISS accepts the following extra options.

   - workingPrefix

     This option specifies the directory for the temporary files used by Invoke Server NAREGISS on remote host.

     If the remote host is a PC cluster, it is recommended to set this option to a directory which is shared by all cluster nodes.

     If omitted, user's home directory is used.

   - CandidateHost

     This option specifies the system on which Ninf-G Executable will run. This is specified by the hostname of the head node of the system.

     Multiple CandidateHost options can be specified.

   - OperatingSystemName

     This option specifies the name of the operating system the computing resources. It is required by NAREGI Super Scheduler.

   - CPUArchitectureName

     This option specifies CPU architecture of the computing resources. It is required by NAREGI Super Scheduler.

   - IndividualCPUCount

     This option specifies minimum number of CPUs per a computing node. This is required by NAREGI Super Scheduler.

   - MemoryLimit

     This option specifies the maximum size of physical memory that the Ninf-G Executable will use. This is required by NAREGI Super Scheduler.

   - logFlags

     This option controls the output of logs of Invoke Server NAREGISS.

     The following values can be specified.

IS_COMMAND	: Output logs about communication between Ninf-G Client and Invoke Server
SS_COMMAND	: Output logs of XML document related NAREGI SS
SS_WF_ID	: Output logs of EPR of NAREGI SS job
ALL	: Output all logs

Multiple values can be specified by delimiting them by spaces.

If omitted, Invoke Server NAREGISS outputs the minimum logs.

Note: If log file is not specified using invoke_server_log attribute in <CLIENT> section or log_filePath in <INVOKE_SERVER> option, this option is ignored.

Note: Whenever a function/object handle is created, Invoke Server receives the Invoke Server options. But this option is effective only at the first time of a handle creation. Therefore, this option must be specified not in invoke_server_option attribute in <SERVER> section but in option attribute in <INVOKE_SERVER> section.

- WallTimeLimit

  This option specifies the maximum job execution wall clock time in seconds. job_maxWallTime attribute of <SERVER> section also specifies the maximum job execution wall clock time, however it is specified in minutes.

  If both this option and job_maxWallTime attribute are specified, the value of this option is used. If neither this option nor job_maxWallTime attribute are specified, "1000 seconds" is used as the default value of the maximum job execution wall clock time.

- MPIType

  This option specifies MPI type.

  If omitted, "GridMPI" is used.

- MPITasksPerHost

  This option specifies the number of processes per host.

  If omitted, "1" is used.

5. Known Problems

   Invoke Server NAREGISS has some problems. Details are described in 11.5 Problems related to NAREGI SS.

4.5 Running the Ninf-G Client program

- Generating the Globus proxy certificate

  ```
  % grid-proxy-init
  ```

  Note: This operation is required if PreWS GRAM or WS GRAM is used.

- Running the Ninf-G Client Program

  ```
  % ./test_client [args ...]
  ```

4.6 Creating application programs

Ninf-G supports the GridRPC API for C and Java.

In this section, the flow of an application program (written in C) for using GridRPC is described and a few typical GridRPC API functions are introduced.

Of the functions described here, those that contain *_np are not included in the GridRPC API standard (i.e., they are specific to Ninf-G).

A full list of the GridRPC APIs and a detailed explanation of each API can be found in chapter 7, "API Reference."

- Flow of an application program for using GridRPC

  The typical flow of an application program for using GridRPC is as follows.

  1. Initialization
  2. Creation of handles
  3. Calling and synchronizing remote functions and remote methods
  4. Destruction of handles
  5. End processing

The functions used in the above processes are described below.

1. Initialization

   The following function is used for initialization.

   ```
   grpc_error_t grpc_initialize(char *configFile)
   ```

   This function accepts the name of the configuration file as an argument, reads the file named by the argument, analyzes the content, and saves the values.

   If the argument value is NULL, the file specified by the NG_CONFIG_FILE environment variable is taken to be the configuration file.

   As the return value, an error status code is returned to inform of failure to read the configuration file or failure to save the values that were read.

   An example of using grpc_initialize() is given below. (In this example, the configuration file name is taken from the command line argument and that value is used as the argument.)

   ```
   main (int argc, char *argv[]) {
          grpc_error_t result;
          char *configFile = argv[1];

          result = grpc_initialize(configFile);
          ...
   ```

2. Creation of handles

   In GridRPC, "handles" are used when performing operations such as executing remote functions and remote methods. A handle must be created before executing a remote function or remote method, but the type of handle created differs with the type of Ninf-G Executable used.

   If only one remote function is defined for the Ninf-G Executable used, a "function handle" is used; if multiple remote methods are defined, an "object handle" is used.

   Functions for creating both kinds of handles are shown below.

   - function handle

     ```
     grpc_error_t
     grpc_function_handle_init(
         grpc_function_handle_t *handle,
         char *server_name,
         char *func_name)
     ```

   - object handle

```
grpc_error_t
grpc_object_handle_init_np(
    grpc_object_handle_t_np *handle,
    char *server_name,
    char *class_name)
```

These functions accept a 'server name' and 'function or class name,' and create a handle for operating the specified Ninf-G Executable on the specified server.

As the return value, an error code is returned to inform of failure to create the handle.

For example, a function handle is created as follows.

```
grpc_function_handle_t *handle;
grpc_function_handle_init(&handle, "server.example.org", "lib/mmul");
```

The following functions for creating multiple handles at one time are also provided by Ninf-G. (See Section 7 for details)

- `grpc_function_handle_array_init_np()`
- `grpc_object_handle_array_init_np()`

3. Calling and synchronizing remote functions and remote methods

   The handle just created can be used to call the specified remote function or remote method on the server. When the call is made, the value of the argument defined by the Ninf-G IDL must be passed.

   The functions used for calling a function differ for a function handle and an object handle. When calling a remote method with an object handle, the name of the remote method must be specified.

   Remote functions and remote methods can be called in two ways, with a 'synchronous call' and with an 'asynchronous call.'

   The synchronous call does not return until the execution of the remote function or remote method is completed.

   The asynchronous call returns either at the beginning or at the completion of the sending of the arguments to the remote function or remote method; it then waits for the completion of the remote function or remote method to obtain the result. (The return timing of the function that makes the asynchronous call can be specified in the configuration file.)

   - Synchronous calling functions

     Functions for making remote functions and remote methods calls of the synchronous type are shown below.

     - function handle

       ```
       grpc_error_t
       grpc_call(
           grpc_function_handle_t *handle, ...)
       ```

     - object handle

       ```
       grpc_error_t
       grpc_invoke_np(
           grpc_object_handle_t_np *handle,
           char *method_name, ...)
       ```

These functions accept the handle and the parameter values to be passed to the remote function or remote method (the remote method name also, in the case of the grpc_invoke_np() function), execute the computation by the specified remote function or remote method, and return as soon as the computation is completed.

As the return value, an error status code is returned to inform the user when the execution of the remote function or remote method fails.

For example, a call to a remote function or remote method defined in the IDL file below is made in the form of grpc_call() below that.

Definition in the IDL file

```
...
mmul(double *A, double *B, double *C)
...
```

Ninf-G Client application program

```
...
double A[10], B[10], C[10];

/* A, B, and C are replaced by values */
...
grpc_function_handle_t *handle;
result = grpc_function_handle_init(&handle,
    "server.example.org", "lib/mmul");
...
result = grpc_call(&handle, A, B, C);
...
```

- Asynchronous calling functions

  Functions for making remote functions and remote methods calls of the asynchronous type are shown below.

  - function handle

```
grpc_error_t
grpc_call_async(
    grpc_function_handle_t *handle,
    grpc_sessionid_t *session_id, ...)
```

  - object handle

```
grpc_error_t
grpc_invoke_async_np(
    grpc_object_handle_t_np *handle,
    char *method_name,
    grpc_sessionid_t *session_id, ...)
```

  These functions accept the handle and the parameter values to be passed to the remote function or remote method (remote method name also, in the case of the grpc_invoke_np() function), issue a request for computation to the specified remote function or remote method, and return when the transmission of arguments begins or when it ends (which can be set in the configuration file).

  If successful, GRPC_NO_ERROR is returned. In the case of an error, an error code is returned.

The returned session ID is used when waiting for the execution results or for other such purposes.

Functions for waiting for the completion of the computation for an asynchronous call are shown below. All of these functions return an error status code to inform of cases in which execution of the session fails.

```
grpc_error_t grpc_wait(grpc_sessionid_t session_id)
```

This waits for completion of the session specified by the session ID passed in the argument and returns when the session ends.

```
grpc_error_t grpc_wait_any(grpc_sessionid_t *id)
```

This waits for completion of any of the current sessions and returns when the session ends.

```
grpc_error_t grpc_wait_and( grpc_sessionid_t *sessions, size_t length)
```

Waits for completion of all of the sessions specified by the array of session IDs and returns when they end.

```
grpc_error_t grpc_wait_or( grpc_sessionid_t *sessions, size_t length, grpc_sessionid_t *id)
```

Waits for completion of any of the sessions specified by the array of session IDs and returns when one of them ends.

```
grpc_error_t grpc_wait_all()
```

This waits for completion of all of the current sessions and returns when they all have ended.

4. Destruction of handles

   For releasing resources, unnecessary "handles" must be destructed. The function for destructing differs with the type of "handles."

   Functions for destructing handles are shown below.

   - function handle

     ```
     grpc_error_t
     grpc_function_handle_destruct(
         grpc_function_handle_t *handle)
     ```

   - object handle

     ```
     grpc_error_t
     grpc_object_handle_destruct_np(
         grpc_object_handle_t_np *handle)
     ```

   These functions destruct the specified handle.

   As the return value, an error status code is returned to inform of failure to destruct the handle.

   If two or more handles were created at once, then the following functions for destructing multiple handles at one time must be used.

   - `grpc_function_handle_array_destruct_np()`
   - `grpc_object_handle_array_destruct_np()`

5. Termination processing

   The following function is used to perform termination processing.

   ```
   grpc_error_t grpc_finalize()
   ```

   This function executes the processing when the Ninf-G Client is terminated.

   The return value is an error status code to inform the user when termination processing fails.

- Other functions

  The API that provides capabilities that have been added in Ninf-G v2 is described below.

  - Callback

    When callback is used, "a function that has both the same name as the name of the callback type argument described in the Ninf-G IDL and the same arguments" must be defined and implemented in the application program.

    Below is an application program that corresponds to the callback example that appears in chapter 3, "Creating and setting up server-side programs"(section 3.1). (Ninf-G Executable and Ninf-G Client exchange status values)

    Note: The maximum number of parameters which can be defined as callback function is 32.

```
/* global */
int executableStatus;
int clientStatus;

void callback_func(int c[], int d[])
{
    executableStatus = c[0];
    d[0] = clientStatus;
}

main()
{
    grpc_function_handle_t handle;
    grpc_error_t result;
    int b;
    ...
    result = grpc_function_handle_init(&handle,
        "server.example.org", "test/callback_test");
    result = grpc_call(&handle, 100, &b, callback_func);
    ...
}
```

  - Checking the session status

    A function for checking the status of a session is shown below.

```
grpc_error_t
grpc_session_info_get_np(
    grpc_sessionid_t session_id,
    grpc_session_info_t_np *info,
    int *status)
```

    This checks the status of the session that corresponds to the session ID specified in the argument.

    When the heartbeat is not obtained normally, GRPC_SESSION_DOWN is returned as the 3rd argument of this function. If an error has occurred, the error code is returned.

- Canceling a session

  A function for canceling a session is shown below.

  ```
  grpc_error_t grpc_cancel(grpc_sessionid_t session_id)
  ```

  This checks the status of the session that corresponds to the session ID specified in the argument.

  An error code is returned as the return value to inform the user that an error has occurred.

appendix

a.1 : How to use multiple user certificates

Ninf-G Client is able to use multiple user proxy certificates. Being enabled by Invoke Server, this capability is useful for using different user proxy certificates according to the security configuration (accepted CAs) of servers.

This section describes how to use multiple certificates.

a.1.1 Create a script which specifies a user proxy certificates used for user authentication by the server. It is recommended to create a script using a template provided by Ninf-G.

Copy the script from template ($NG_DIR/etc/ng_invoke_server.GTtempl).

```
% cp $NG_DIR/etc/ng_invoke_server.GTtempl $NG_DIR/bin/ng_invoke_server.GT4cert1
% chmod u+x $NG_DIR/bin/ng_invoke_server.GT4cert1
```

Modify the copied script in which you have to specify the user proxy certificate(*1) and the script file(*2) which you will use.

```
#! /bin/sh

X509_USER_PROXY=/path/to/x509up_xxxx <- (*1)
export X509_USER_PROXY
exec $NG_DIR/bin/ng_invoke_server.GT4py <- (*2)
```

a.1.2 Modify the client configuration file

Modify the client configuration file and specify the Invoke Server that you created at a.1.1.

```
<SERVER>
    hostname  example.org
        :
    invoke_server  GT4cert1
</SERVER>
```

a.2 : How to implement cascading RPC

Ninf-G4 supports cascading RPC, which enables Ninf-G Executable to call GridRPC API. Cascading RPC is realized by (1) implementing remote functions that calls GridRPC API (server-side implementation) and (2) configuring Ninf-G client to enable delegation of full-proxy certificates (client-side configuration).

1. Server-side implementation

   In the IDL file,

- Set `ng_cc` to Compiler and Linker.

  Example:

  ```
  Compiler "$(NG_DIR)/bin/ng_cc";
  Linker "$(NG_DIR)/bin/ng_cc";
  ```

- Include the grpc.h on IDL file.

  Example:

  ```
  Globals { #include <grpc.h> }
  ```

- Implement a remote function that calls GridRPC API.

  It is implemented by embedding GridRPC APIs such as grpc_initialize(), grpc_function_handle_init(), and grpc_call() in the body of the remote function.

- Compile the IDL file by an ordinary way.

2. Client-side configuration
   - Set the "`delegate_full_proxy true`" option to Invoke Server GT4py in <SERVER> section of Ninf-G Client Configuration file.

     Example:

     ```
     <SERVER>
     hostname ...
     invoke_server GT4py
     invoke_server_option "delegate_full_proxy true"
     </SERVER>
     ```

   - NG_DIR environment variable must be set in Ninf-G Executable if Invoke Server is used for subordinate RPC since Invoke Server requires NG_DIR environment variable. NG_DIR environment variable can be set by either "environment" attribute in <SERVER> section of the Client Configuration file on the client side or "path" attribute in <INVOKE_SERVER> section of the Client Configuration file on the remote side.

     Example:

     ```
     <SERVER>
     ...
     environment NG_DIR=/remote/server/ng_dir/path
     </SERVER>
     ```

   - Some notes about working directory of Ninf-G Executable.

     Ninf-G Executable searches the following files in the current working directory.

     - Client Configuration file which is passed as an argument to grpc_initialize().
     - Local LDIF files specified by the Client Configuration file.

     If staging is off, Ninf-G Executable runs on the directory in which the Ninf-G Executable exists.

     If staging is on, Ninf-G Executable always runs on the user's home directory unless working directory is explicitly specified by the user using "workDirectory" attribute in the Client Configuration file.

Example:

```
<SERVER>
...
workDirectory /path/to/work/directory/of/executable
</SERVER>
```

Note: Cascading RPC is available for Ninf-G Version 4.2.0 or later.

---

last update : $Date: 2008/09/12 08:27:42 $

5. Examples

This section give you a tutorial of how to use the Ninf-G system for programming on the Grid. Simplicity of programming is the most beneficial aspect of the Ninf-G system, and we hope that users will be able to gridify his programs easily after reading this document. We hope to extend this example further to cover more advanced Ninf-G features. Examples are provided for GRPC API.

Grid RPC API

Gridifying a Numerical Library with Grid RPC API

We first cover the simple case where the library to be Gridifyied is defined as a linkable library function. Below is a sample code of a simple matrix multiply. The first scalar argument specifies the size of the matrix (n by n), parameters `a` and `b` are references to matrices to be multiplied, and c is the reference to the result matrix. Notice that, 1) the matrix (defined as arrays) do not itself embody size as type information, and 2) as a result there is a dependency between n and a, b, c. In fact, since array arguments are passed as a reference, one must assume the contents of the array are implicitly shared by the caller and the callee, with arbitrary choices as to using them as input, output, or temporary data structures.

```
void mmul(int n, double * a, double * b, double * c)
{
    double t;
    int i, j, k;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            t = 0;
            for (k = 0; k < n; k++){
                t += a[i * n + k] * b[k * n + j];
            }
            c[i * n + j] = t;
        }
    }
}
```

The main routine which calls mmul() might be as follows:

```
main()
{
    double A[N*N], B[N*N], C[N*N];

    initMatA(N, A); /* initialize */
    initMatB(N, B); /* initialize */

    mmul(N, A, B, C);
}
```

In order to "Gridify", or more precisely, allow mmul to be called remotely via GridRPC, we must describe the interface of the function so that information not embodied in the language type system becomes sufficiently available to the GridRPC system to make the remote call. Although future standardization is definitely conceivable, currently each GridRPC system has its own IDL (Interface Description Language); for example, Ninf has its own NinfIDL definition. Below we give the interface of mmul() defined by the NinfIDL syntax:

```
1: Module mmul;
2:
3: Define mmul(IN int N, IN double A[N*N],
4:             IN double B[N*N], OUT double C[N*N])
5: "matmul"
6: Required "mmul_lib.o"
7: Calls "C" mmul(N, A, B, C);
```

Line 1 declares the module name to be defined. There is a one-to-one correspondence between a module and an IDL file, and each module can have multiple entries to gridify multiple functions. Lines 3-7 are the definition for a particular entry mmul/mmul. Here, lines 3 and 4 declare the interface of the entry. The difference between a NinfIDL entry definition and the C prototype definition is that there are no return values (the return value of the Ninf call is used to return status info), argument input/output modes are specified, and array sizes are described in terms of the scalar arguments.

We note here that NinfIDL has special features to efficiently support gridifying of a library (similar features are found in Netsolve IDL). In contrast to standard procedure calls within a shared memory space, GridRPC needs to transfer data over the network. Transferring the entire contents of the array will be naturally very costly, especially for huge matrices appearing in real applications. Here, one will quickly observe that surprising number of numerical libraries take for granted the fact that address space of data structures, in particular arrays are shared, and (a) only use subarrays of the passed arrays, (b) write back results in the passed arrays, and (c) pass arrays as scratchpad temporaries. The portion of the arrays to be operated, etc., are determined by the semantics of the operation according to the input parameters passed to the function. For example in mmul, the whole arrays need to be passed, and their sizes are all N by N, where N is the first scalar parameter; A and B only need to be passed as input parameters and their contents do not change, while C is used as a return argument and thus need not be shipped to the server, but the result needs to be shipped back to the client. In general, in order to determine and minimize the size of transfer, NinfIDL allows flexible description of the array shape used by the remote library. One can specify leading dimensions, subarrays, and strides. In fact arbitrary arithmetic expressions involving constants and scalar arguments can be used in the array size expressions.

Line 5 is the comment describing the entry, while line 6 specifies the necessary object file when the executable for the particular file is to be linked. Line 7 gives the actual library function to be called, and the calling sequence; here "C" denotes C-style (row-major) array layout.

The user compiles this IDL file using the Ninf IDL compiler, and generates the stub code and its makefile. By executing this makefile a Ninf executable is generated. The user will subsequently register the executable to the server using the registry tool.

Now the client us ready to make the call of the network. In order to make a GridRPC call, the user modifies his original main program in the following manner. We notice that only the function call is modified---No need to change the program to adjust to the skeleton that the IDL generator generates as is with typical RPC systems such as CORBA. Moreover, we note that the IDL, the stub files and the executables are only resident on the server side, and the client only needs to link his program with a generic Ninf client library.

```
main()
{
    double A[N*N], B[N*N], C[N*N];
    grpc_function_handle_t handle;

    grpc_initialize(argv[1]);

    initMatA(N, A); /* initialize */
    initMatB(N, B); /* initialize */

    grpc_function_handle_default(&handle, "mmul/mmul");

    if (grpc_call(&handle, N, A, B, C) != GRPC_NO_ERROR) {
        fprintf(stderr, "Error in grpc_call¥n");
        exit(1);
    }

    grpc_function_handle_destruct(&handle);
    ...
    grpc_finalize();
}
```

Gridifying Programs that use Files

The above example assumes that the numerical routine is supplied as a library with well-defined function API, or at least its source is available in a way such that it could easily converted into a library. In practice, many numerical routines are only available in a non-library executable and/or binary form, with input/output interfaces using files. In order to gridify such "canned" applications, GridRPC systems typically support remote files and their automatic management/transfer.

We take gnuplot as an example. Gnuplot in non-interactive mode inputs script from a specified file, and outputs the resulting graph to the standard output. Below is an example gnuplot script.

```
set terminal postscript
set xlabel "x"
set ylabel "y"
plot f(x) = sin(x*a), a = .2, f(x), a = .4, f(x)
```

If this script is saved under a filename "gplot":

```
> gnuplot gplot > graph.ps
```

will store the postscript representation of the graph to the file graph.ps. In order to execute gnuplot remotely, we must package it appropriately, and moreover must automatically transfer the input (gplot) and output (graph.ps) files between the client and the server.

Ninf-G IDL provides a type *filename* to specify that the particular argument is a file. Below is an example of using gnuplot via GridRPC.

```
Module plot;
Define plot(IN filename plotfile, OUT filename psfile )
"invoke gnuplot"
{
    char buffer[1000];
    sprintf(buffer, "gnuplot %s > %s", plotfile, psfile);
    system(buffer);
}
```

The IDL writes the string command sequence to invoke gnuplot into a variable buffer[], and invokes gnuplot as a system library. The file specified as an input is automatically transferred to the temporary directory of the server, and its temporary file name is passed to the stub function. As for the output file, only the temporary file name is created and passed to the stub function. After the stub program is executed, the files in output mode as specified in the IDL are automatically transferred to the client, and saved there under the name given in the argument.

Below is an example of how this function might be called via GridRPC.

```
#include <stdio.h>
#include "grpc.h"

main(int argc, char **argv)
{
    grpc_function_handle_t handle;

    grpc_initialize(argv[1]);

    grpc_function_handle_default(&handle, "plot/plot");

    if (grpc_call(&handle, argv[2], argv[3]) != GRPC_NO_ERROR) {
        fprintf(stderr, "Error in grpc_call¥n");
        exit(1);
    }
    grpc_function_handle_destruct(&handle);
    ...
    grpc_finalize();
}
```

We also note that, by combining this feature with the technique of using multiple servers simultaneously described in the next section, we can process large amount of data at once.

Using Multiple Servers for Parallel Programming on the Grid --- The Parameter Sweep Survey Example.

GridRPC can serve as a task-parallel programming abstraction, whose programs can scale from local workstations to the Grid. Here, we take an example of simple parameter sweep survey, and investigate how it can be easily programmed using GridRPC.

Calculating PI using a simple Monte Carlo Method

As an example, we compute the value of PI using a simple Monte Carlo Method. We generate a large number of random points within the square region that exactly encloses a unit circle (actually, 1/4 of a circle). We calculate the value of PI by inverse computing the area of the circle according to the probability that the points will fall within the circle. The program below shows the original sequential version.

```
long pi_trial(int seed, long times)
{
    long l, long counter = 0;

    srandom(seed);
    for (l = 0; l < times; l++){
        double x = (double)random() / RAND_MAX;
        double y = (double)random() / RAND_MAX;
        if (x * x + y * y < 1.0)
            counter++;
    }
    return counter;
}

main(int argc, char **argv)
{
    double pi;
    long times = atol(argv[1]);
    count = pi_trial(10, times);
    pi = 4.0 * (count / (double) times);
    printf("PI = %f¥n", pi);
}
```

Gridifying the PI program.

First, we rewrite the program so that it does the appropriate GridRPC calls. The following steps are needed:

1. Separate out the pi_trail() function into a separate file (say, trial_pi.c), and create its object file trial_pi.o using a standard C compiler.
2. Describe the interface of pi_trial in an IDL file.

```
Module pi;

Define pi_trial(IN int seed, IN long times, OUT long * count)
"monte carlo pi computation"
Required "pi_trial.o"
{
    long counter;
    counter = pi_trial(seed, times);
    *count = counter;
}
```

3. Rewrite the main program so that it makes a GridRPC call.

```
main(int argc, char **argv)
{
    double pi;
    long times, count;
    grpc_function_handle_t handle;

    grpc_initialize(argv[1]);

    times = atol(argv[2]);

    grpc_function_handle_default(&handle, "pi/pi_trial");

    if (grpc_call(&handle, 10, times, &count) != GRPC_NO_ERROR) {
        fprintf(stderr, "Failed in grpc_call¥n");
        exit(2);
    }

    pi = 4.0 * ( count / (double) times);
    printf("PI = %f¥n", pi);

    grpc_function_handle_destruct(&handle);

    grpc_finalize();
}
```

We now have made the body of the computation remote. The next phase is to parallelise it.

Employing Multiple Servers for Task Parallel Computation.

We next rewrite the main program so that parallel tasks are distributed to multiple servers. Although distribution of tasks are possible using metaserver scheduling with Ninf (and Agents with Netsolve), it is sometimes better to specify a host explicitly for performance reasons, for low overhead and explicit load balancing. Ninf-G allows explicit specification of servers by specifying the hostname in the initialization of the function handle.

The standard grpc_call() RPC is synchronous in that the client waits until the completion of the computation on the server side. For task-parallel execution, Ninf-G facilitates several asynchronous call APIs. For example, the most basic asynchronous call grpc_call_async is almost identical to grpc_call except that it returns immediately after all the arguments have been sent. The return value is the session ID of the asynchronous call; the ID is used for various synchronizations such as waiting for the return value of the call.

There are several calls for synchronization. The most basic is the `grpc_wait(grpc_sessionid_t ID)`, where we wait for the result of the asynchronous call with the supplied session ID. grpc_wait_all() waits for all preceding asynchronous invocations made. Here, we employ grpc_wait_all() to parallelize the above PI client so that it uses multiple simultaneous remote server calls:

```
#include "grpc.h"
#define NUM_HOSTS 8
char * hosts[] = {"node0.example.org", "node1.example.org", "node2.example.org", "node3.example.org",
                  "node4.example.org", "node5.example.org", "node6.example.org", "node7.example.org"};

grpc_function_handle_t handles[NUM_HOSTS];
grpc_sessionid_t ids[NUM_HOSTS];

main(int argc, char **argv){
    double pi;
    long times, count[NUM_HOSTS], sum;
    char * config_file;
    int i;
    if (argc < 3) {
        fprintf(stderr, "USAGE: %s CONFIG_FILE TIMES ¥n", argv[0]);
        exit(2);
    }
    config_file = argv[1];
    times = atol(argv[2]) / NUM_HOSTS;

21      /* Initialize GRPC runtimes. */
22      if (grpc_initialize(config_file) != GRPC_NO_ERROR) {
23          exit(2);
24      }
25      /* Initialize handles. */
26      for (i = 0; i < NUM_HOSTS; i++)
27          grpc_function_handle_init(&handles[i], hosts[i], "pi/pi_trial");
28
29      for (i = 0; i < NUM_HOSTS; i++) {
30          /* Parallel non-blocking remote function invocation. */
31          if (grpc_call_async(&handles[i], &ids[i], i, times, &count[i]) != GRPC_NO_ERROR){
32              grpc_perror_np("pi_trial");
33              exit(2);
34          }
35      }
36      /* Sync. */
37      if (grpc_wait_all() != GRPC_NO_ERROR) {
38          grpc_perror_np("wait_all");
39          exit(2);
40      }
41
42      for (i = 0; i < NUM_HOSTS; i++)
43          grpc_function_handle_destruct(&handles[i]);
44
45      /* Compute and display pi. */
46      for (i = 0, sum = 0; i < NUM_HOSTS; i++)
47          sum += count[i];
48      pi = 4.0 * ( sum / ((double) times * NUM_HOSTS));
49      printf("PI = %f¥n", pi);
50
51      /* Finalize GRPC runtimes. */
52      grpc_finalize();
53 }
```

We specify the number of server hosts and their names in lines 2 and 3-4, respectively. Line 6 is the port number used, and line 19 divides the number of trials with the number of servers, determining the number of trials per server. The for loop in lines 29-35 invokes the servers asynchronously. Line 47 aggregates the results returned from all the servers.

In this manner, we can easily write a parallel parameter sweep survey program using the task parallel primitives of GridRPC. We next modify the program to perform dynamic load balancing.

---

last update : $Date: 2005/07/07 11:06:40 $

6 Ninf-G IDL Specifications

Ninf-G IDL files are processed by CPP(C Pre Processor).
Please pay attention to CPP keywords(ex. "#include").

6.1 Ninf-G IDL keywords

Keywords described below are reserved by Ninf-G. You can't use these words as the name of module, functions or parameters.

- `Module` module_name ;

  This specifies the identifier of the Ninf-G Executable. The name specified here is used for naming the Ninf-G stub file and the makefile for compiling it, etc.

- `Globals` { ... C descriptions ... }

  Global variables that can be used from all of the remote functions and remote methods that can be executed by the Ninf-G Executable can be defined. The expression between braces ({ c }) is written in C.

  Note: The maximum number of Globals which can be defined in one IDL file is 100.

  Note: The strings written in "Globals" are put to the Ninf-G stub file which is generated by ng_gen as it is. So you can put CPP keywords (like "#include < foo.h >") to the Ninf-G stub file by using "Globals".
  But if you want multiple of them, please use multiple "Globals" like below.

  ```
  Globals{#include < foo.h >}
  Globals{#include < bar.h >}
  ```

  You can't write like below because Ninf-G IDL files are processed by CPP.

  ```
  Globals{
  #include < foo.h >
  #include < bar.h >
  }
  ```

- `Compiler " ... ";`

  This specifies the compiler to be used when the Ninf-G Executable is compiled.

- `CompileOptions " ... ";`

  This specifies the options to be applied when the Ninf-G Executable is compiled.

  Note: The maximum number of CompileOptions which can be defined in one IDL file is 100.

- `Linker " ... ";`

  This specifies the linker to be used at link time of Ninf-G Executable.

- `Library " ... ";`

This specifies the libraries and the options to be applied at the link time of Ninf-G Executable.

For example, ("-lxxx") and the object file (".o") or library file (".a") will be specified in this section. The options specified in this section are applied only at link time not at compile time (*.c -> *.o).

Note: The maximum number of Library which can be defined in one IDL file is 100.

- `FortranFormat "...";`

  The Ninf-G stub is output as a C program. When functions that are written in FORTRAN are called from that program, the name must be converted to a particular format ("_%s_", where %s is a function name). That conversion format is shown below.

  The following character strings have special meanings; they are replaced by particular character strings at processing time.

  - `%s` : Original function name
  - `%l` : All-uppercase function name

- `DefClass` class_name

  ["description"]
  [`Required` "files-or-libs"]
  [`Backend "MPI" | "BLACS"`]
  [`Language "C" | "C++" | "FORTRAN"`]
  [`Shrink "yes" | "no"`]
  `{ [DefState{ ... }]`
  `DefMethod` method_name (arg1, arg2, ...) { ...}
  ... }

  Defines a remote class.

  One `DefState` section and one or more remote methods can be defined for the Ninf-G Executable. The `DefState` section can be omitted.

  The attribute values that can be set are described below.

description	A character string that explains this Ninf-G Executable.
	Any character string may be used.
`Required`	The remote function object file (".o").
	This specifies the object file (".o") that stores the functions used by the remote function.
`Backend`	This specifies the backend at run time.
	"MPI" or "BLACS" can be specified. If omitted, no backend is used.
`Language`	The programming language in which the remote method is written.
	"C", "C++" or "FORTRAN" can be specified. If omitted, the default value is "C".
`Shrink`	Specify whether or not shrinking is enabled.
	If "yes" is specified, shrinking is applied to the array elements. If omitted, shrinking is not done.

- `DefState { ...` C descriptions ... `}`

  The variables maintained by Ninf-G Executable are defined in C.

  Those values can be shared by multiple remote methods of the same Ninf-G Executable.

- `DefMethod`

  ["description"]
  [`CalcOrder` exp]
  { { C descriptions } | `Calls` lang-spec function_name (arg1, arg2, ...); }

  Defines a remote method.

  A C program can be written in the remote method, and functions can be called from within the program or by using the `Calls` keyword to specify the functions to be called.

  Note: following keywords are reserved by Ninf-G. So you can't use these keywords in your remote function.

  - label : `"ng_stub_end"`, `"ng_stub_mpi_end"`
  - variable : `"ng_stub_rank"`

  The language in which the function that is called from the remote method is written can be specified in the lang-spec argument that comes right after the `Calls` keyword. Currently, `"C"`, `"C++"` or `"FORTRAN"` can be specified. If omitted, the default `"C"` is used.
  If `"FORTRAN"` is specified here, the called function must be converted to the format specified by `FortranFormat` before it is called.

  The attribute values that can be set are described below.

"description"	A character string that explains this remote method.
	Any character string can be used.
`CalcOrder`	Calculation order.
	The cost of executing the remote method can be specified here in arbitrary expression form.

  - Special methods

    If argumentless remote methods are defined with the following names, those remote methods will be executed automatically when the Ninf-G Executable starts up and ends.

`_initialize()`	Executed at startup
`_finalize()`	Executed at end

- `Define` function_name (parameter1, parameter2, ...)

  ["description"]
  [`Required` "files-or-libs"]
  [`CalcOrder` exp]
  [`Backend "MPI" | "BLACS"`]
  [`Language "C" | "C++" | "FORTRAN"`]
  [`Shrink "yes" | "no"`]
  { { C descriptions } | `Calls` lang-spec function_name (arg1, arg2, ...); }

  Defines a remote function.

  This remote function is a method that is exclusive to the Ninf-G Executable. A C program can be written in the remote function, and functions can be called from within the program or by using the `Calls` keyword to specify the functions to be called.

  Note: following keywords are reserved by Ninf-G. So you can't use these keywords in your remote function.

  - label : "ng_stub_end", "ng_stub_mpi_end"
  - variable : "ng_stub_rank"

The language in which the function that is called from the remote method can be specified in the lang-spec argument that comes right after the Calls keyword. Currently, "C", "C++" or "FORTRAN" can be specified.

If omitted, the default "C" is used. If "FORTRAN" is specified here, the called function must be converted to the format specified by FortranFormat before it is called.

The attribute values that can be set are described below.

"description"	A character string that explains this remote function.
	Any character string can be used.
Required	The remote function object file (".o").
	This specifies the object file (".o") that stores the functions used by the remote function.
CalcOrder	Calculation order.
	The cost of executing the remote function can be specified here in arbitrary expression form.
Backend	This specifies the backend at run time.
	"MPI" or "BLACS" can be specified. If omitted, no backend is used.
Language	The programming language in which the remote function is written.
	"C", "C++" or "FORTRAN" can be specified. If omitted, the default value is "C".
Shrink	Specify whether or not shrinking is enabled.
	If "yes" is specified, shrinking is applied to the array elements. If omitted, shrinking is not done.

- Parameters

  The arguments passed to the remote function or remote method are written with the following syntax.

  [mode-spec] [type-spec] parameter_name [[dimension[:range]]+

  Note: A scalar variable cannot be used for the parameter of callback function.

  Note: The maximum number of the parameter which can be defined as callback function is 32.

"mode-spec"	parameter mode
	IN, OUT, INOUT or WORK can be specified. "allocate" and "broadcast" are also available as modifiers for specifying a method for parameter distribution in MPI and BLACS backends. < parameter mode > IN : Client -> Server OUT : Server -> Client INOUT: Both "IN" and "OUT" WORK : The values are not transmitted, but an area is reserved on the server side. < parameter mode modifier > allocate : allocates a memory area for the variable on every node whose rank is not 0. This specifier can be specified with all mode ("IN", "INOUT", "OUT", "WORK").

	`broadcast` : allocates a memory area for the variable on every node whose rank is not 0. The value of the variable is broadcasted from the rank0 node. This specifier can be specified with "IN" and "INOUT" mode. Note: The mode cannot be specified for Callback. A scalar variable cannot be used for the parameter that specifies the `OUT` mode. Note: "allocate" and "broadcast" can be written before or after the mode keywords("IN", "INOUT", "OUT", "WORK"). If "allocate" and "broadcast" are omitted, nothing will be done. Note: "allocate" and "broadcast" are ignored if the backend of the remote method is neither "MPI" nor "BLACS".
"type-spec"	Parameter type
	`char`, `short`, `int`, `long`, `float`, `double`, `string`, `scomplex`, `dcomplex` or `filename` can be specified. The `char`, `short`, `int`, `long`, `float` and `double` terms correspond to the respective C types. Note: The `string` term corresponds to `char []`. Callback does not specify the type. Note: There is the example program for filename type variable in Ninf-G package. (< package directory > /diag/file_test)
parameter_name	any character string
Array specification	The specification of an array of parameters
	If the parameter is an array, it is written as "[" + expression + "]" and its size is specified. A multidimensional array can be defined with multiple "[" + expression + "]". Also, the variables used by the argument can be used in the expression. It is also possible to specify the starting position, ending position and shrinking interval within "[" + expression + "]" as shown below to define arrays. "[" + expression (size) + ":" + expression (start position) + "]" "[" + expression (size) + ":" + expression (start position) + "," + expression (ending position) + "]" "[" + expression (size) + ":" + expression (start position) + "," + expression (ending position) + "," + expression (shrinking interval) + "]" If a shrinking interval is specified and "Shrink" is enabled, shrinking of variable values is done for transmission and saving. Note: If "starting positon" is omitted, then it's set to the value of "0". Note: If "ending positon" is omitted or it's specified "0", then it's set to the value of "size". Note: If "shrinking interval" is omitted or it's specified "0", then it's set to the value of "1". Note: Shrinking of string type variable is not supported.

- Expressions

  Expressions can be used to specify the calculation order and the size of parameter arrays.

  It can include constants, other `IN` mode scalar parameter in the function definition, and some operators. We provide some operators (`+`, `-`, `*`, `/`, `%`...). Priority among these operators is same as ANSI C. You can also use parentheses in expressions.

Examples)

Arithmetic operations	10 / 2 * 5
Three term calculation	n % 10 ? 5: 2

6.2 Ninf-G IDL syntax

```
/* program toplevel */
program:  /* empty */
      | declaration_list
    ;

declaration_list:
        declaration
      | declaration_list declaration
    ;

declaration:
        "Module" IDENTIFIER ';'
      | "CompileOptions" STRING ';'
      | "Globals" globals_body
      | "Library" STRING ';'
      | "FortranFormat" STRING ';'
      | "FortranStringConvention" IDENTIFIER ';'
      | "Define" interface_definition opt_string option_list interface_body
      | "DefClass" IDENTIFIER opt_string defclass_option_list '{' define_list '}'
      | "Compiler" STRING ';'
      | "Linker" STRING ';'
    ;

define_list:
        /* empty */
      | define_item
      | define_list define_item
    ;

define_item:
        "DefState" '{'
      | "DefMethod" interface_definition opt_string
      defmethod_option_list interface_body
    ;

defmethod_option_list:
        /* empty */
      | defmethod_option
      | defmethod_option_list defmethod_option
    ;

defmethod_option:
        calcorder
    ;

defclass_option_list:
        /* empty */
      | defclass_option
      | defclass_option_list defclass_option
    ;

defclass_option:
        required
      | backend
      | shrink
      | language
    ;

option_list:
        /* empty */
      | decl_option
      | option_list decl_option
    ;

decl_option:
        required
      | backend
      | shrink
      | calcorder
      | language
    ;

interface_definition:
        IDENTIFIER '(' parameter_list ')'
      | IDENTIFIER '(' parameter_callback_list ')'
      | IDENTIFIER '(' callback_list ')'
    ;

callback_list:
        callback
      | callback_list ',' callback
    ;

parameter_callback_list:
        parameter_list ',' callback
      | parameter_callback_list ',' callback
    ;

parameter_list:
        /* empty */
      | parameter
      | parameter_list ',' parameter
    ;

callback:
        IDENTIFIER '(' parameter_list ')'
    ;

parameter:
        decl_specifier declarator
    ;

decl_specifier:
        mode_specifier type_specifier
      | type_specifier mode_specifier
      | type_specifier mode_specifier type_specifier
    ;

type_specifier:
        TYPE
      | TYPE TYPE
      | TYPE TYPE TYPE
    ;

mode_specifier:
        MODE
      | MODE DISTMODE
      | DISTMODE MODE
    ;

declarator:
        IDENTIFIER
      | '(' declarator ')'
      | declarator '['expr_or_null ']'
      | declarator '['expr_or_null ':' range_spec ']'
      | '*' declarator
    ;

range_spec:
        expr     /* upper limit */
      | expr ',' expr     /* lower limit and upper limit */
      | expr ',' expr ',' expr     /* lower, upper and step */
    ;

opt_string:
        /* empty */
      | STRING
    ;

required:
        "Required" STRING
    ;

backend:
        "Backend" STRING
    ;

shrink:
        "Shrink" STRING
    ;

language:
        "Language" STRING
    ;

calcorder:
        "CalcOrder" expr
    ;

interface_body:
    '{'  /* C statements */ '}'
    | "Calls" opt_string IDENTIFIER '(' id_list ')' ';'
    ;

globals_body:
    '{'  /* C statements */ '}'
    ;

id_list: IDENTIFIER
    | id_list ',' IDENTIFIER
    | /* empty */
    ;


/* index description */
expr_or_null:
    expr
    | /* empty */
    ;

expr:
     unary_expr
    | expr '/' expr
    | expr '%' expr
    | expr '+' expr
    | expr '-' expr
    | expr '*' expr
    | expr '^' expr
    | expr '<'  expr
    | expr "<=" expr
    | expr ">'  expr
    | expr ">=" expr
    | expr "!=" expr
    | expr "==" expr
    | expr '?' expr ':' expr
    ;

unary_expr:
      primary_expr
    | '*' expr
    | '-' expr
    ;

primary_expr:
     primary_expr '[' expr ']'
    | IDENTIFIER
    | CONSTANT
    | '('  expr  ')'
    ;

/* TYPE = int, char, short, long, long float, double string scomplex dcomplex filename */
/* MODE = IN, OUT, INOUT, WORK */
/* DISTMODE = allocate broadcast */
/* IDENTIFIER = name */
/* CONSTANT = integer literals, floating point literals */
/* STRING = "..." */
```

6.3 IDL sample

- Function version (one method only)

  idl-sample/function/sample.idl

- Object version (multiple methods)

  idl-sample/object/sample_obj.idl

- samples from diagnostic program in Ninf-G package.

  - data types
  - filename type
  - callback type
  - shrink arguments/results
  - session cancel

- MPI sample

  - PI with MPI

    "PI" is a sample program in Ninf-G package.

  - data types test with MPI

- samples from users.

  Followings are IDL files used by Ninf-G users.

  - users-sample1.idl
  - users-sample2.idl
  - users-sample3.idl
  - users-sample4.idl

6.4 Example of output results

- sample.idl processing results(IDL sample 1 from section 6.3)

  idl-sample/function/sample.mak(Makefile)
  idl-sample/function/_stub_sin.c
  idl-sample/function/_stub_mmul.c
  idl-sample/function/_stub_mmul2.c
  idl-sample/function/_stub_FFT.c

- sample_obj.idl processing results (IDL sample 2 from section 6.3)

  idl-sample/object/sample_object.mak(Makefile)
  idl-sample/object/_stub_sample_object.c

---

last update : $Date: 2007/07/10 09:17:42 $

7. GridRPC API Reference Manual

This section lists and gives you information about all Ninf-G API.

API Index

Ninf-G Client initialization and finalization API

- grpc_initialize()
- grpc_finalize()
- grpc_config_file_read_np()

Ninf-G Handle API

type	operation	array	default	attr	API
function	initialize				grpc_function_handle_init()
function	initialize			attr	grpc_function_handle_init_with_attr_np()
function	initialize		default		grpc_function_handle_default()
function	destruct				grpc_function_handle_destruct()
function	initialize	array			grpc_function_handle_array_init_np()
function	initialize	array		attr	grpc_function_handle_array_init_with_attr_np()
function	initialize	array	default		grpc_function_handle_array_default_np()
function	destruct	array			grpc_function_handle_array_destruct_np()
function	get_attr				grpc_function_handle_get_attr_np()
function	get_handle				grpc_function_handle_get_from_session_np()
function	get_handle				grpc_get_handle()
function	perror				grpc_function_handle_perror_np()
function	get_error				grpc_function_handle_get_error_np()
function	is_ready				grpc_function_handle_is_ready_np()
object	initialize				grpc_object_handle_init_np()
object	initialize			attr	grpc_object_handle_init_with_attr_np()
object	initialize		default		grpc_object_handle_default_np()
object	destruct				grpc_object_handle_destruct_np()
object	initialize	array			grpc_object_handle_array_init_np()
object	initialize	array		attr	grpc_object_handle_array_init_with_attr_np()
object	initialize	array	default		grpc_object_handle_array_default_np()
object	destruct	array			grpc_object_handle_array_destruct_np()

object	get_attr	grpc_object_handle_get_attr_np()
object	get_handle	grpc_object_handle_get_from_session_np()
object	perror	grpc_object_handle_perror_np()
object	get_error	grpc_object_handle_get_error_np()
object	is_ready	grpc_object_handle_is_ready_np()

type	operation	API
attribute	initialize	grpc_handle_attr_initialize_np()
attribute	destruct	grpc_handle_attr_destruct_np()
attribute	get	grpc_handle_attr_get_np()
attribute	set	grpc_handle_attr_set_np()
attribute	release	grpc_handle_attr_release_np()

Ninf-G Session Invocation API

type	attr	arg_stack	async	API
function				grpc_call()
function			async	grpc_call_async()
function		arg_stack		grpc_call_arg_stack()
function		arg_stack	async	grpc_call_arg_stack_async()
function	attr			grpc_call_with_attr_np()
function	attr		async	grpc_call_async_with_attr_np()
function	attr	arg_stack		grpc_call_arg_stack_with_attr_np()
function	attr	arg_stack	async	grpc_call_arg_stack_async_with_attr_np()
object				grpc_invoke_np()
object			async	grpc_invoke_async_np()
object		arg_stack		grpc_invoke_arg_stack_np()
object		arg_stack	async	grpc_invoke_arg_stack_async_np()
object	attr			grpc_invoke_with_attr_np()
object	attr		async	grpc_invoke_async_with_attr_np()
object	attr	arg_stack		grpc_invoke_arg_stack_with_attr_np()
object	attr	arg_stack	async	grpc_invoke_arg_stack_async_with_attr_np()

type	operation	API
attribute	initialize	grpc_session_attr_initialize_np()
attribute	initialize	grpc_session_attr_initialize_with_object_handle_np()
attribute	destruct	grpc_session_attr_destruct_np()
attribute	get	grpc_session_attr_get_np()
attribute	set	grpc_session_attr_set_np()
attribute	release	grpc_session_attr_release_np()

Ninf-G Argument Stack API

- grpc_arg_stack_new()
- grpc_arg_stack_destruct()
- grpc_arg_stack_push_arg()
- grpc_arg_stack_pop_arg()

Ninf-G Session Wait API

- grpc_wait()
- grpc_wait_all()
- grpc_wait_any()
- grpc_wait_and()
- grpc_wait_or()

Ninf-G Session Control API

- grpc_cancel()
- grpc_cancel_all()
- grpc_probe()
- grpc_probe_or()
- grpc_session_info_get_np()
- grpc_session_info_release_np()
- grpc_session_info_set_threshold_np()
- grpc_session_info_remove_np()
- grpc_get_error()

Ninf-G Errors API

- grpc_error_string()
- grpc_perror_np()
- grpc_get_failed_sessionid()
- grpc_last_error_get_np()

Ninf-G Others API

- grpc_signal_handler_set_np()

Ninf-G Server side API

- grpc_is_canceled_np()

For Backward Compatibility API (Not recommend)

- grpc_get_info_np()
- grpc_get_last_info_np()
- grpc_get_last_error_np()

---

last update : $Date: 2006/02/13 08:06:40 $

NAME

`grpc_initialize` - Initializes Ninf-G.

SYNOPSIS

`grpc_error_t grpc_initialize(char *config_file_name)`

ARGUMENTS

`char *config_file_name`
: The configuration file for the client

DESCRIPTION

The grpc_initialize() function initializes Ninf-G and the Globus Toolkit used by Ninf-G. The configuration file specified by config_file_name is read and the values are saved in global variables within Ninf-G.

If NULL or an empty string is specified in config_file_name , the configuration file specified by the NG_CONFIG_FILE environment variable is used as the configuration file.

If the NG_CONFIG_FILE environment variable is also undefined or an empty string, then $HOME/.ngconf is used as the configuration file.

Signal

In its implementation, Ninf-G Client uses SIGINT, SIGTERM and SIGHUP. When a Ninf-G Client catches one of them, it cancels all outstanding sessions, destructs all function/object handles, then exits.

Attention

grpc_initialize() overwrites signal handlers of signals used by Ninf-G Client and grpc_finalize() restores them.

Ninf-G Client may not work correctly if signal() or sigaction() system call is called between grpc_initialize() and grpc_finalize(). Instead, grpc_signal_handler_set_np() should be used for registering signal handlers.

Ninf-G Client compiled with pthread flavor may not work correctly if a thread is created prior to grpc_initialize() or some signals are removed from signal mask in each thread.

This function is MT-unsafe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_CONFIGFILE_NOT_FOUND`
: The configuration file does not exist.
: The configuration file could not be read.

`GRPC_CONFIGFILE_ERROR`
: The content of the configuration file is invalid.

`GRPC_ALREADY_INITIALIZED`
: Ninf-G is already initialized.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

last update : $Date: 2006/02/22 03:08:14 $

NAME

`grpc_finalize` - Executes the Ninf-G end processing.

SYNOPSIS

```
grpc_error_t grpc_finalize()
```

DESCRIPTION

The grpc_finalize() function performs the processing for terminating Ninf-G.

All outstanding jobs are canceled and resources used by Ninf-G are released. The cancellation procedure includes end processing for the Globus Toolkit.

This function is MT-unsafe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/10/25 02:01:04 $

NAME

`grpc_config_file_read_np` - Reads the configuration file.

SYNOPSIS

`grpc_error_t grpc_config_file_read_np(char *config_file_name)`

ARGUMENTS

`char *config_file_name`
    The configuration file for the client

DESCRIPTION

The grpc_config_file_read_np() function reads the configuration file and changes the configuration of the running environment of the Ninf-G Client dynamically.

This function can be used for dynamic addition of the information about new computing resources which were not known at the startup time.

Once this function is called, the new configuration is effective for newly created function/object handles, i.e. existing handles keep their old configuration.

This function does not updates <CLIENT> sections. In each section, if an attribute value is not specified, the default value is used.

As same as grpc_initialize() function, if NULL or an empty string is specified in config_file_name, the configuration file specified by the NG_CONFIG_FILE environment variable is used as the configuration file.

If the NG_CONFIG_FILE environment variable is also undefined or an empty string, then $HOME/.ngconf is used as the configuration file.

This function is MT-safe.

Note: No information will be discarded by this function. Only the addition and modification of the configuration are performed.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, an error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
    GRPC client is not initialized yet.
`GRPC_CONFIGFILE_NOT_FOUND`
    The configuration file does not exist.
    The configuration file could not be read.
`GRPC_CONFIGFILE_ERROR`
    The content of the configuration file is invalid.
`GRPC_OTHER_ERROR_CODE`
    Internal error detected.

last update : $Date: 2005/07/11 08:13:35 $

NAME

`grpc_function_handle_init` - Initializes a function handle.

SYNOPSIS

```
grpc_error_t grpc_function_handle_init( grpc_function_handle_t *handle, char *server_name, char *func_name)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle to be initialized

`char *server_name`
: The host name (resource manager contact) of the remote machine.

`char *func_name`
: The function to be executed on the remote machine

DESCRIPTION

The grpc_function_handle_init() function initializes a function handle.

Every Globus Toolkit GRAM resource manager contact can be specified as a server_name argument. Resource manager contact can be one of the followings:

host name
host name:port number
host name:port number/jobmanager
host name/jobmanager
host name:/jobmanager
host name::subject
host name:port number:subject
host name/jobmanager:subject
host name:/jobmanager:subject
host name:port number/jobmanager:subject

If a user defines tag name in <SERVER> section, tag name can be specified in the host name of the Resource Manager Contact in server_name argument.

The API searches corresponding tag name in a client configuration file. If the tag name is not found, the API searches corresponding server name. The first match will be selected if multiple sections have the same host name.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`

GRPC client cannot find any server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the default server.

`GRPC_RPC_REFUSED`
: GRPC server refused the initialization.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2006/01/16 09:39:26 $

NAME

`grpc_function_handle_init_with_attr_np` - grpc_function_handle_init_with_attr_np - Initializes a function handle.

SYNOPSIS

```
grpc_error_t grpc_function_handle_init_with_attr_np( grpc_function_handle_t *handle, grpc_handle_attr_t_np
*attr)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle to be initialized

`grpc_handle_attr_t_np *attr`
: The attributes of the handle

DESCRIPTION

The grpc_function_handle_init_with_attr_np() function initializes a function handle.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find any server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the default server.

`GRPC_RPC_REFUSED`
: GRPC server refused the initialization.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/10/25 02:01:04 $

NAME

`grpc_function_handle_default` - Initializes a function handle.

SYNOPSIS

```
grpc_error_t grpc_function_handle_default( grpc_function_handle_t *handle, char *func_name)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle to be initialized

`char *func_name`
: The function to be executed on the remote machine

DESCRIPTION

The grpc_function_handle_default() function initializes a function handle.
For the remote host name, the default remote host name specified in the configuration file is used.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find any server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the default server.

`GRPC_RPC_REFUSED`
: GRPC server refused the initialization.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/10/25 02:01:04 $

NAME

`grpc_function_handle_destruct` - Destructs a function handle.

SYNOPSIS

```
grpc_error_t grpc_function_handle_destruct(grpc_function_handle_t *handle)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle to be destructed

DESCRIPTION

The grpc_function_handle_destruct() function destructs a function handle.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_FUNCTION_HANDLE`
: Function handle specified by handle is invalid.

`GRPC_TIMEOUT_NP`
: Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_function_handle_array_init_np` - Initializes a function handle.

SYNOPSIS

```
grpc_error_t grpc_function_handle_array_init_np( grpc_function_handle_t *handles, size_t nhandles, char
*server_name, char *func_name)
```

ARGUMENTS

`grpc_function_handle_t *handles`
: The function handle to be initialized

`size_t nhandles`
: The number of function handles to be initialized

`char *server_name`
: The host name (resource manager contact) of the remote machine.

`char *func_name`
: Function to be executed on the remote machine

DESCRIPTION

The grpc_function_handle_array_init_np() function initializes the function handles.

The server_name argument can be specified as the same way with server_name argument in grpc_function_handle_init().

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find any server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the default server.

`GRPC_RPC_REFUSED`
: GRPC server refused the initialization.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/10/25 02:01:04 $

NAME

`grpc_function_handle_array_init_with_attr_np` - Initializes a function handle.

SYNOPSIS

```
grpc_error_t grpc_function_handle_array_init_with_attr_np( grpc_function_handle_t *handles, size_t nhandles,
grpc_handle_attr_t_np *attr)
```

ARGUMENTS

`grpc_function_handle_t *handles`
: The function handle to be initialized

`size_t nhandles`
: The number of function handles to be initialized

`grpc_handle_attr_t_np *attr`
: The handle attributes

DESCRIPTION

The grpc_function_handle_array_init_with_attr() initializes a function handle.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find any server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the default server.

`GRPC_RPC_REFUSED`
: GRPC server refused the initialization.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/10/25 02:01:04 $

NAME

`grpc_function_handle_array_default_np` - Initializes function handle.

SYNOPSIS

```
grpc_error_t grpc_function_handle_array_default_np( grpc_function_handle_t *handles, size_t nhandles, char
*func_name)
```

ARGUMENTS

`grpc_function_handle_t *handles`
: The function handle to be initialized

`size_t nhandles`
: The number of function handles to be initialized

`char *func_name`
: The function to be executed on the remote host

DESCRIPTION

The grpc_function_handle_default_np() function initializes the function handle.
For the remote machine host name, the default remote host name specified in the configuration file is used.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find any server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the default server.

`GRPC_RPC_REFUSED`
: GRPC server refused the initialization.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/10/25 02:01:04 $

NAME

`grpc_function_handle_array_destruct_np` - Destructs a function handle.

SYNOPSIS

```
grpc_error_t grpc_function_handle_array_destruct_np( grpc_function_handle_t *handles, size_t nhandles)
```

ARGUMENTS

`grpc_function_handle_t *handles`
: The function handle to be destructed

`size_t nhandles`
: The number of function handles to be initialized

DESCRIPTION

The grpc_function_handle_array_destruct_np() function destructs a function handle.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_FUNCTION_HANDLE`
: Function handle specified by handle is invalid.

`GRPC_TIMEOUT_NP`
: Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_function_handle_get_attr_np` - Gets handle attributes.

SYNOPSIS

```
grpc_error_t grpc_function_handle_get_attr_np( grpc_function_handle_t *handle, grpc_handle_attr_t_np *attr)
```

ARGUMENTS

`grpc_function_handle_t *handle`
  The function handle
`grpc_handle_attr_t_np *attr`
  The handle attributes

DESCRIPTION

The grpc_function_handle_get_attr_np() function returns handle attributes for the handle.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
  GRPC client is not initialized yet.
`GRPC_INVALID_FUNCTION_HANDLE`
  Function handle specified by handle is invalid.
`GRPC_OTHER_ERROR_CODE`
  Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_function_handle_get_from_session_np` - Gets a function handle.

SYNOPSIS

```
grpc_error_t grpc_function_handle_get_from_session_np( grpc_function_handle_t **handle, grpc_sessionid_t
session_id)
```

ARGUMENTS

`grpc_function_handle_t **handle`
: The function handle

`grpc_sessionid_t session_id`
: The session ID

DESCRIPTION

The grpc_function_handle_get_from_session_np() function returns a function handle for the specified session ID.

If the value specified in session_id is ID of session of Object Handle, this API returns error.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_SESSION_ID`
: Session ID is not valid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_get_handle` - Gets a function handle.

SYNOPSIS

```
grpc_error_t grpc_get_handle(grpc_function_handle_t **handle, grpc_sessionid_t session_id)
```

ARGUMENTS

`grpc_function_handle_t **handle`
: The function handle

`grpc_sessionid_t session_id`
: The session ID

DESCRIPTION

The grpc_get_handle() function returns a function handle for the session specified by the session ID.

If the value specified in session_id is ID of session of Object Handle, this API returns error.

In Ninf-G, there are grpc_function_handle_get_from_session_np() which gets a function handle, and grpc_object_handle_get_from_session_np() which gets an object handle.

The function of grpc_get_handle() and grpc_function_handle_get_from_session_np() is the same.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_SESSION_ID`
: Session ID is not valid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_function_handle_perror_np` - Displays an error.

SYNOPSIS

```
grpc_error_t grpc_function_handle_perror_np(grpc_function_handle_t *handle, char *str)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle

`char *str`
: The character string to be displayed

DESCRIPTION

The grpc_function_handle_perror_np() function displays the function handle error in standard output.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_FUNCTION_HANDLE`
: Function handle specified by handle is invalid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_function_handle_get_error_np` - Returns the error code generated by the handle.

SYNOPSIS

```
grpc_error_t grpc_function_handle_get_error_np(grpc_function_handle_t *handle)
```

ARGUMENTS

`grpc_function_handle_t *handle`
    The function handle

DESCRIPTION

The grpc_function_handle_get_error_np() function returns the error code that was generated in the handle specified by function handle.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
    GRPC client is not initialized yet.
`GRPC_INVALID_FUNCTION_HANDLE`
    Function handle specified by handle is invalid.
`GRPC_OTHER_ERROR_CODE`
    Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_function_handle_is_ready_np` - Checks whether the specified handle is ready or not

SYNOPSIS

```
grpc_error_t grpc_function_handle_is_ready_np(grpc_function_handle_t *handle)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle

DESCRIPTION

The grpc_function_handle_is_ready_np() function returns whether the specified handle is ready or not.

This function is MT-safe.

RETURN VALUE

If the function handle is ready, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_FUNCTION_HANDLE`
: Function handle specified by handle is invalid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_object_handle_init_np` - grpc_object_handle_init_np - Initializes an object handle.

SYNOPSIS

```
grpc_error_t grpc_object_handle_init_np( grpc_object_handle_t_np *handle, char *server_name, char
*class_name)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle to be initialized

`char *server_name`
: The host name (resource manager contact) of the remote machine.

`char *class_name`
: The class to be executed on the remote machine

DESCRIPTION

The grpc_object_handle_init_np() function initializes an object handle.

The server_name argument can be specified as the same way with server_name argument in grpc_function_handle_init().

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find any server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the default server.

`GRPC_RPC_REFUSED`
: GRPC server refused the initialization.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/10/25 02:01:04 $

NAME

`grpc_object_handle_init_with_attr_np` - Initializes an object handle.

SYNOPSIS

```
grpc_error_t grpc_object_handle_init_with_attr_np( grpc_object_handle_t_np *handle, grpc_handle_attr_t_np
*attr)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle to be initialized

`grpc_handle_attr_t_np *attr`
: The handle attributes

DESCRIPTION

The grpc_object_handle_init_with_attr_np() function initializes an object handle.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find any server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the default server.

`GRPC_RPC_REFUSED`
: GRPC server refused the initialization.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/10/25 02:01:04 $

NAME

`grpc_object_handle_default_np` - grpc_object_handle_default_np - Initializes an object handle.

SYNOPSIS

```
grpc_error_t grpc_object_handle_default_np(grpc_object_handle_t_np *handle, char *class_name)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle to be initialized

`char *class_name`
: The class to be executed on the remote machine

DESCRIPTION

The grpc_object_handle_default_np() function initializes object handles.
For the remote machine host name, the default remote host name specified in the configuration file is used.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find any server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the default server.

`GRPC_RPC_REFUSED`
: GRPC server refused the initialization.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/10/25 02:01:04 $

NAME

`grpc_object_handle_destruct_np` - grpc_object_handle_destruct_np - Destructs an object handle

SYNOPSIS

```
grpc_error_t grpc_object_handle_destruct_np(grpc_object_handle_t_np *handle)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle to be destructed

DESCRIPTION

The grpc_object_handle_destruct_np() function destructs an object handle.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_OBJECT_HANDLE_NP`
: Object handle specified by handle is invalid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_object_handle_array_init_np` - grpc_object_handle_array_init_np - Initializes an object handle.

SYNOPSIS

```
grpc_error_t grpc_object_handle_array_init_np( grpc_object_handle_t_np *handles, size_t nhandles, char
*server_name, char *class_name)
```

ARGUMENTS

`grpc_object_handle_t_np *handles`
: The object handle to be initialized

`size_t nhandles`
: The number of object handles to be initialized

`char *server_name`
: The host name (resource manager contact) of the remote machine.

`char *class_name`
: The class to be executed on the remote machine

DESCRIPTION

The grpc_object_handle_array_init_np() initializes the object handles.

The server_name argument can be specified as the same way with server_name argument in grpc_function_handle_init().

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find any server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the default server.

`GRPC_RPC_REFUSED`
: GRPC server refused the initialization.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/10/25 02:01:04 $

NAME

`grpc_object_handle_array_init_with_attr_np` - grpc_object_handle_array_init_with_attr_np - Initializes an object handle.

SYNOPSIS

```
grpc_error_t grpc_object_handle_array_init_with_attr_np( grpc_object_handle_t_np *handles, size_t nhandles,
grpc_handle_attr_t_np *attr)
```

ARGUMENTS

`grpc_object_handle_t_np *handles`
: The object handle to be initialized

`size_t nhandles`
: The number of object handles to be initialized

`grpc_handle_attr_t_np *attr`
: The handle attributes

DESCRIPTION

The grpc_object_handle_array_init_with_attr_np() function initializes an object handle.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find any server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the default server.

`GRPC_RPC_REFUSED`
: GRPC server refused the initialization.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/10/25 02:01:04 $

NAME

`grpc_object_handle_array_default_np` - grpc_object_handle_array_default_np - Initializes an object handle.

SYNOPSIS

```
grpc_error_t grpc_object_handle_array_default_np( grpc_object_handle_t_np *handles, size_t nhandles, char
*class_name)
```

ARGUMENTS

`grpc_object_handle_t_np *handles`
: The object handle to be initialized

`size_t nhandles`
: The number of object handles to be initialized

`char *class_name`
: The class to be executed on the remote machine

DESCRIPTION

The grpc_object_handle_array_default_np() function initializes an object handle.
For the remote machine host name, the default remote host name specified in the configuration file is used.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find any server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the default server.

`GRPC_RPC_REFUSED`
: GRPC server refused the initialization.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/10/25 02:01:04 $

NAME

`grpc_object_handle_array_destruct_np` - grpc_object_handle_array_destruct_np - Destructs an object handle.

SYNOPSIS

`grpc_error_t grpc_object_handle_array_destruct_np( grpc_object_handle_t_np *handles, size_t nhandles)`

ARGUMENTS

`grpc_object_handle_t_np *handles`
: The object handle to be destructed

`size_t nhandles`
: The number of object handles to be initialized

DESCRIPTION

The grpc_object_handle_array_destruct_np()function destructs an object handle.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_OBJECT_HANDLE_NP`
: Object handle specified by handle is invalid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_object_handle_get_attr_np` - Gets handle attributes.

SYNOPSIS

```
grpc_error_t grpc_object_handle_get_attr_np( grpc_object_handle_t_np *handle, grpc_handle_attr_t_np *attr)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle

`grpc_handle_attr_t_np *attr`
: The handle attributes

DESCRIPTION

The grpc_object_handle_get_attr_np() function returns handle attributes for the handle.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_OBJECT_HANDLE_NP`
: Object handle specified by handle is invalid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_object_handle_get_from_session_np` - Gets an object handle.

SYNOPSIS

```
grpc_error_t grpc_object_handle_get_from_session_np( grpc_object_handle_t_np **handle, grpc_sessionid_t
session_id)
```

ARGUMENTS

`grpc_object_handle_t_np **handle`
: The object handle

`grpc_sessionid_t session_id`
: The session ID

DESCRIPTION

The grpc_object_handle_get_from_session_np()function returns the object handle for the session specified by the session ID.

If the value specified in session_id is ID of session of Function Handle, this API returns error.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_SESSION_ID`
: Session ID is not valid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_object_handle_perror_np` - Displays the error.

SYNOPSIS

```
grpc_error_t grpc_object_handle_perror_np(grpc_object_handle_t_np *handle, char *str)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle

`char *str`
: The character string to be displayed

DESCRIPTION

The grpc_object_handle_perror_np() function displays the error for the object handle in the standard error output.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_OBJECT_HANDLE_NP`
: Object handle specified by handle is invalid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_object_handle_get_error_np` - Returns the error code generated by the handle.

SYNOPSIS

```
grpc_error_t grpc_object_handle_get_error_np(grpc_object_handle_t_np *handle)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle

DESCRIPTION

The grpc_object_handle_get_error_np() function returns the error code that was generated in the handle specified by object handle.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_OBJECT_HANDLE_NP`
: Object handle specified by handle is invalid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_object_handle_is_ready_np` - Checks whether the specified handle is ready or not

SYNOPSIS

```
grpc_error_t grpc_object_handle_is_ready_np(grpc_object_handle_t_np *handle)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle

DESCRIPTION

The grpc_object_handle_is_ready_np() function returns whether the specified handle is ready or not.

This function is MT-safe.

RETURN VALUE

If the object handle is ready, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_OBJECT_HANDLE_NP`
: Object handle specified by handle is invalid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_handle_attr_initialize_np` - Initializes function handle attributes.

SYNOPSIS

```
grpc_error_t grpc_handle_attr_initialize_np(grpc_handle_attr_t_np *attr)
```

ARGUMENTS

`grpc_handle_attr_t_np *attr`
: The function handle attributes to be initialized

DESCRIPTION

The grpc_handle_attr_initialize_np() function initializes function handle attributes.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_handle_attr_destruct_np` - Destructs handle attributes.

SYNOPSIS

```
grpc_error_t grpc_handle_attr_destruct_np(grpc_handle_attr_t_np *attr)
```

ARGUMENTS

`grpc_handle_attr_t_np *attr`
: The handle attributes to be destructed

DESCRIPTION

The grpc_handle_attr_destruct_np() function destructs handle attributes.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_handle_attr_get_np` - Gets handle attributes.

SYNOPSIS

```
grpc_error_t grpc_handle_attr_get_np(grpc_handle_attr_t_np *attr, grpc_handle_attr_name_t_np name, void
**value)
```

ARGUMENTS

`grpc_handle_attr_t_np *attr`
: The handle attributes

`grpc_handle_attr_name_t_np name`
: The names of the attributes to be get

`void *value`
: The values to be get

DESCRIPTION

The grpc_handle_attr_get_np() function returns the values of handle attributes.

See the manual of grpc_handle_attr_set_np() for details of grpc_handle_attr_name_t_np.

Memory area allocated for the attribute value should be released by calling grpc_handle_attr_release_np() after the value was referred.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/08/04 02:32:49 $

NAME

grpc_handle_attr_set_np - Sets handle attributes.

SYNOPSIS

grpc_error_t grpc_handle_attr_set_np(grpc_handle_attr_t_np *attr, grpc_handle_attr_name_t_np name, void *value)

ARGUMENTS

grpc_handle_attr_t_np *attr
: The handle attributes

grpc_handle_attr_name_t_np name
: The names of the attributes to be set

void *value
: The values to be set

```
typedef enum {                           /* Format for passing to value */
    GRPC_HANDLE_ATTR_HOSTNAME,          /* char */
    GRPC_HANDLE_ATTR_PORT,              /* int */
    GRPC_HANDLE_ATTR_JOBMANAGER,        /* char */
    GRPC_HANDLE_ATTR_SUBJECT,           /* char */
    GRPC_HANDLE_ATTR_FUNCNAME,          /* char */
    GRPC_HANDLE_ATTR_JOBSTARTTIMEOUT,   /* int */
    GRPC_HANDLE_ATTR_JOBSTOPTIMEOUT,    /* int */
    GRPC_HANDLE_ATTR_WAIT_ARG_TRANSFER, /* grpc_argument_transfer_t_np */
    GRPC_HANDLE_ATTR_QUEUENAME,         /* char */
    GRPC_HANDLE_ATTR_MPI_NCPUS,         /* int */
} grpc_handle_attr_name_t_np;
```

- GRPC_HANDLE_ATTR_HOSTNAME
- GRPC_HANDLE_ATTR_PORT
- GRPC_HANDLE_ATTR_JOBMANAGER
- GRPC_HANDLE_ATTR_SUBJECT
- GRPC_HANDLE_ATTR_FUNCNAME

  Those are the same as the arguments of the existing grpc_function_handle_init() function.

- GRPC_HANDLE_ATTR_JOBSTARTTIMEOUT

  This specifies the time-out value in seconds for when a job is started.

  When the grpc_call() function *1 is called and the job has not started, if the time specified by job_timeout elapses after the job start-up request was issued, then grpc_call() ends with a time-out error.

  If the value 0 is specified, the process continues to wait for the job to start without timing out.

- GRPC_HANDLE_ATTR_JOBSTOPTIMEOUT

  The time out time for job completion is specified in seconds.

  If the -1 value is specified, there is no time-out and the process waits until the job stops. If the 0 value is specified, the process doesn't wait for the job to stop.

- GRPC_HANDLE_ATTR_WAIT_ARG_TRANSFER

  This flag specifies whether or not to wait for the transfer of arguments in an asynchronous RPC.

  The default is to wait for the transfer.

  The value set up with this attribute is shown below.

    - `GRPC_ARGUMENT_TRANSFER_WAIT`

      It waits for the end of transfer argument.

    - `GRPC_ARGUMENT_TRANSFER_NOWAIT`

      It does not wait for the end of transfer argument.

    - `GRPC_ARGUMENT_TRANSFER_COPY`

      The copy of an argument is made.

- `GRPC_HANDLE_ATTR_QUEUENAME`

  Target the GRAM job to a queue (class) name as defined by the scheduler at the defined (remote) resource.

- `GRPC_HANDLE_ATTR_MPI_NCPUS`

  This specifies the number of CPUs for MPI function.

DESCRIPTION

The grpc_handle_attr_set_np() function sets the values of handle attributes.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2008/03/12 06:52:30 $

NAME

`grpc_handle_attr_release_np` - frees memory for the handle attribute value.

SYNOPSIS

```
grpc_error_t grpc_handle_attr_release_np(void *value)
```

ARGUMENTS

`void *value`
: Pointer to the value obtained by grpc_handle_attr_get_np()

DESCRIPTION

The grpc_handle_attr_release_np() frees memory for the value obtained by grpc_handle_attr_get_np().

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, an error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/08/04 02:32:49 $

NAME

`grpc_call` - Executes an RPC.

SYNOPSIS

```
grpc_error_t grpc_call(grpc_function_handle_t *handle, ...)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle

`Other arguments`
: Arguments that are passed to the function called by RPC

DESCRIPTION

The grpc_call() function calls the function defined by the function handle.
The grpc_call() function is blocked until the called function completes execution.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find the specified server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the specified server.

`GRPC_INVALID_FUNCTION_HANDLE`
: Function handle specified by handle is invalid.

`GRPC_RPC_REFUSED`
: RPC invocation was refused by the server, possibly because of a security issue.

`GRPC_COMMUNICATION_FAILED`
: Communication with the server failed somehow.

`GRPC_TIMEOUT_NP`
: Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_call_async` - Executes an RPC.

SYNOPSIS

```
grpc_error_t grpc_call_async( grpc_function_handle_t *handle, grpc_sessionid_t *session_id, ...)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle

`grpc_sessionid_t *session_id`
: The session ID

`Other arguments`
: The arguments passed to the function called by the RPC

DESCRIPTION

The grpc_call_async() calls the function defined by the function handle.
The grpc_call_async() does not wait for completion of the called function.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find the specified server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the specified server.

`GRPC_INVALID_FUNCTION_HANDLE`
: Function handle specified by handle is invalid.

`GRPC_RPC_REFUSED`
: RPC invocation was refused by the server, possibly because of a security issue.

`GRPC_COMMUNICATION_FAILED`
: Communication with the server failed somehow.

`GRPC_TIMEOUT_NP`
: Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/11/18 08:19:13 $

NAME

`grpc_call_arg_stack` - Executes an RPC.

SYNOPSIS

```
int grpc_call_arg_stack(grpc_function_handle_t *handle, grpc_arg_stack_t *stack)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle

`grpc_arg_stack_t *stack`
: The stack that holds the arguments for passing to the function called by the RPC.

DESCRIPTION

The grpc_call_arg_stack() function calls the function defined by the function handle.
The grpc_call_arg_stack() function is blocked until the called function completes execution.

This function is MT-safe.

RETURN VALUE

If successful, 0 is returned. In the case of an error, -1 is returned

ERRORS

`GRPC_NOT_INITIALIZED`
: The grpc_initialize() function has not been executed.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_call_arg_stack_async` - Executes an RPC.

SYNOPSIS

```
int grpc_call_arg_stack_async(grpc_function_handle_t *handle, grpc_arg_stack_t *stack)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle

`grpc_arg_stack_t *stack`
: The stack for holding the arguments to be passed to the function called by the RPC

DESCRIPTION

The grpc_call_arg_stack_async() function calls the function defined by function handle.
The grpc_call_arg_stack() function does not wait for the execution of the called function to complete.

This function is MT-safe.

RETURN VALUE

If successful, the session ID is returned. In the case of an error, -1 is returned

ERRORS

`GRPC_NOT_INITIALIZED`
: The grpc_initialize() function has not been executed.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_call_with_attr_np` - Executes an RPC.

SYNOPSIS

```
grpc_error_t grpc_call_with_attr_np(grpc_function_handle_t *handle, grpc_session_attr_t_np *session_attr,
...)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle

`grpc_session_attr_t_np *session_attr`
: The attributes of the session

`Other arguments`
: Arguments that are passed to the function called by RPC

DESCRIPTION

The grpc_call_with_attr_np() function calls the function defined by the function handle.
The grpc_call_with_attr_np() function is blocked until the called function completes execution.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find the specified server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the specified server.

`GRPC_INVALID_FUNCTION_HANDLE`
: Function handle specified by handle is invalid.

`GRPC_RPC_REFUSED`
: RPC invocation was refused by the server, possibly because of a security issue.

`GRPC_COMMUNICATION_FAILED`
: Communication with the server failed somehow.

`GRPC_TIMEOUT_NP`
: Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_call_async_with_attr_np` - Executes an RPC.

SYNOPSIS

```
grpc_error_t grpc_call_async_with_attr_np( grpc_function_handle_t *handle, grpc_sessionid_t *session_id,
grpc_session_attr_t_np *session_attr, ...)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle

`grpc_sessionid_t *session_id`
: The session ID

`grpc_session_attr_t_np *session_attr`
: The attributes of the session

`Other arguments`
: The arguments passed to the function called by the RPC

DESCRIPTION

The grpc_call_async_with_attr_np() calls the function defined by the function handle.
The grpc_call_async_with_attr_np() does not wait for completion of the called function.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find the specified server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the specified server.

`GRPC_INVALID_FUNCTION_HANDLE`
: Function handle specified by handle is invalid.

`GRPC_RPC_REFUSED`
: RPC invocation was refused by the server, possibly because of a security issue.

`GRPC_COMMUNICATION_FAILED`
: Communication with the server failed somehow.

`GRPC_TIMEOUT_NP`
: Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/11/18 08:19:13 $

NAME

`grpc_call_arg_stack_with_attr_np` - Executes an RPC.

SYNOPSIS

```
int grpc_call_arg_stack_with_attr_np(grpc_function_handle_t *handle, grpc_session_attr_t_np *session_attr,
grpc_arg_stack_t *stack)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle

`grpc_session_attr_t_np *session_attr`
: The attributes of the session

`grpc_arg_stack_t *stack`
: The stack that holds the arguments for passing to the function called by the RPC.

DESCRIPTION

The grpc_call_arg_stack_with_attr_np() function calls the function defined by the function handle.
The grpc_call_arg_stack_with_attr_np() function is blocked until the called function completes execution.

This function is MT-safe.

RETURN VALUE

If successful, 0 is returned. In the case of an error, -1 is returned

ERRORS

`GRPC_NOT_INITIALIZED`
: The grpc_initialize() function has not been executed.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_call_arg_stack_async_with_attr_np` - Executes an RPC.

SYNOPSIS

```
int grpc_call_arg_stack_async_with_attr_np(grpc_function_handle_t *handle, grpc_session_attr_t_np
*session_attr, grpc_arg_stack_t *stack)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle

`grpc_session_attr_t_np *session_attr`
: The attributes of the session

`grpc_arg_stack_t *stack`
: The stack for holding the arguments to be passed to the function called by the RPC

DESCRIPTION

The grpc_call_arg_stack_async_with_attr_np() function calls the function defined by function handle.
The grpc_call_arg_stack_with_attr_np() function does not wait for the execution of the called function to complete.

This function is MT-safe.

RETURN VALUE

If successful, the session ID is returned. In the case of an error, -1 is returned

ERRORS

`GRPC_NOT_INITIALIZED`
: The grpc_initialize() function has not been executed.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_invoke_np` - Executes an RPC.

SYNOPSIS

```
grpc_error_t grpc_invoke_np(grpc_object_handle_t_np *handle, char *method_name, ...)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle

`char *method_name`
: The method name

`Other arguments`
: Arguments to be passed to the function called by RPC

DESCRIPTION

The grpc_invoke_np() function calls the method defined by the object handle.
The grpc_invoke_np() function is blocked until execution of the called method is completed.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find the specified server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the specified server.

`GRPC_INVALID_OBJECT_HANDLE_NP`
: Object handle specified by handle is invalid.

`GRPC_RPC_REFUSED`
: RPC invocation was refused by the server, possibly because of a security issue.

`GRPC_COMMUNICATION_FAILED`
: Communication with the server failed somehow.

`GRPC_TIMEOUT_NP`
: Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_invoke_async_np` - Executes an RPC.

SYNOPSIS

```
grpc_error_t grpc_invoke_async_np( grpc_object_handle_t_np *handle, char *method_name, grpc_sessionid_t
*session_id, ...)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle

`char *method_name`
: The method name

`Other arguments`
: Arguments to be passed to the function called by RPC

DESCRIPTION

The grpc_invoke_async_np() function calls the method defined by the object handle.
The grpc_invoke_async_np() function does not wait for the called method to complete execution.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find the specified server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the specified server.

`GRPC_INVALID_FUNCTION_HANDLE`
: Function handle specified by handle is invalid.

`GRPC_RPC_REFUSED`
: RPC invocation was refused by the server, possibly because of a security issue.

`GRPC_COMMUNICATION_FAILED`
: Communication with the server failed somehow.

`GRPC_TIMEOUT_NP`
: Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/11/18 08:19:13 $

NAME

`grpc_invoke_arg_stack_np` - Executes an RPC.

SYNOPSIS

```
int grpc_invoke_arg_stack_np(grpc_object_handle_t_np *handle, char *method_name, grpc_arg_stack_t *stack)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle

`char *method_name`
: The method name

`grpc_arg_stack_t *stack`
: The stack for storing arguments to be passed to the function called in the RPC

DESCRIPTION

The grpc_invoke_arg_stack_np() function calls the method defined by the object handle.
The grpc_invoke_arg_stack_np() function is blocked until the execution of the called method is completed.

This function is MT-safe.

RETURN VALUE

If successful, 0 is returned. In the case of an error, -1 is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: The grpc_initialize() function has not been executed.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_invoke_arg_stack_async_np` - Executes an RPC.

SYNOPSIS

```
int grpc_invoke_arg_stack_async_np(grpc_object_handle_t_np *handle, char *method_name, grpc_arg_stack_t
*stack)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle

`char *method_name`
: The method name

`grpc_arg_stack_t *stack`
: The stack for storing arguments to be passed to the function called in the RPC

DESCRIPTION

The grpc_invoke_arg_stack_async_np() function calls the method defined by the object handle.
The grpc_invoke_arg_stack_async_np() function does not wait for the called method to complete execution.

This function is MT-safe.

RETURN VALUE

If successful, the session ID is returned. In the event of an error, -1 is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: The grpc_initialize() function has not been executed.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_invoke_with_attr_np` - Executes an RPC.

SYNOPSIS

```
grpc_error_t grpc_invoke_with_attr_np(grpc_object_handle_t_np *handle, char *method_name,
grpc_session_attr_t_np *session_attr, ...)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle

`char *method_name`
: The method name

`grpc_session_attr_t_np *session_attr`
: The attributes of the session

`Other arguments`
: Arguments to be passed to the function called by RPC

DESCRIPTION

The grpc_invoke_with_attr_np() function calls the method defined by the object handle.
The grpc_invoke_with_attr_np() function is blocked until execution of the called method is completed.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find the specified server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the specified server.

`GRPC_INVALID_OBJECT_HANDLE_NP`
: Object handle specified by handle is invalid.

`GRPC_RPC_REFUSED`
: RPC invocation was refused by the server, possibly because of a security issue.

`GRPC_COMMUNICATION_FAILED`
: Communication with the server failed somehow.

`GRPC_TIMEOUT_NP`
: Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_invoke_async_with_attr_np` - Executes an RPC.

SYNOPSIS

```
grpc_error_t grpc_invoke_async_with_attr_np( grpc_object_handle_t_np *handle, char *method_name,
grpc_sessionid_t *session_id, grpc_session_attr_t_np *session_attr, ...)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle

`char *method_name`
: The method name

`grpc_sessionid_t *session_id`
: The session ID

`grpc_session_attr_t_np *session_attr`
: The attributes of the session

`Other arguments`
: Arguments to be passed to the function called by RPC

DESCRIPTION

The grpc_invoke_async_with_attr_np() function calls the method defined by the object handle.
The grpc_invoke_async_with_attr_np() function does not wait for the called method to complete execution.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SERVER_NOT_FOUND`
: GRPC client cannot find the specified server.

`GRPC_FUNCTION_NOT_FOUND`
: GRPC client cannot find the function on the specified server.

`GRPC_INVALID_FUNCTION_HANDLE`
: Function handle specified by handle is invalid.

`GRPC_RPC_REFUSED`
: RPC invocation was refused by the server, possibly because of a security issue.

`GRPC_COMMUNICATION_FAILED`
: Communication with the server failed somehow.

`GRPC_TIMEOUT_NP`
: Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/11/18 08:19:13 $

NAME

`grpc_invoke_arg_stack_with_attr_np` - Executes an RPC.

SYNOPSIS

```
int grpc_invoke_arg_stack_with_attr_np(grpc_object_handle_t_np *handle, char *method_name,
grpc_session_attr_t_np *session_attr, grpc_arg_stack_t *stack)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle

`char *method_name`
: The method name

`grpc_session_attr_t_np *session_attr`
: The attributes of the session

`grpc_arg_stack_t *stack`
: The stack for storing arguments to be passed to the function called in the RPC

DESCRIPTION

The grpc_invoke_arg_stack_with_attr_np() function calls the method defined by the object handle.
The grpc_invoke_arg_stack_with_attr_np() function is blocked until the execution of the called method is completed.

This function is MT-safe.

RETURN VALUE

If successful, 0 is returned. In the case of an error, -1 is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: The grpc_initialize() function has not been executed.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_invoke_arg_stack_async_with_attr_np` - Executes an RPC.

SYNOPSIS

```
int grpc_invoke_arg_stack_async_with_attr_np(grpc_object_handle_t_np *handle, char *method_name,
grpc_session_attr_t_np *session_attr, grpc_arg_stack_t *stack)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The object handle

`char *method_name`
: The method name

`grpc_session_attr_t_np *session_attr`
: The attributes of the session

`grpc_arg_stack_t *stack`
: The stack for storing arguments to be passed to the function called in the RPC

DESCRIPTION

The grpc_invoke_arg_stack_async_with_attr_np() function calls the method defined by the object handle. The grpc_invoke_arg_stack_async_with_attr_np() function does not wait for the called method to complete execution.

This function is MT-safe.

RETURN VALUE

If successful, the session ID is returned. In the event of an error, -1 is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: The grpc_initialize() function has not been executed.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_session_attr_initialize_np` - Initializes session attributes with function handle.

SYNOPSIS

```
grpc_error_t grpc_session_attr_initialize_np(grpc_function_handle_t *handle, grpc_session_attr_t_np *attr)
```

ARGUMENTS

`grpc_function_handle_t *handle`
: The function handle

`grpc_session_attr_t_np *attr`
: The session attributes to be initialized

DESCRIPTION

The grpc_session_attr_initialize_np() function initializes session attributes.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_FUNCTION_HANDLE`
: Function handle specified by handle is invalid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_session_attr_initialize_with_object_handle_np` - Initializes session attributes with object handle.

SYNOPSIS

```
grpc_error_t grpc_session_attr_initialize_with_object_handle_np(grpc_object_handle_t_np *handle,
grpc_session_attr_t_np *attr)
```

ARGUMENTS

`grpc_object_handle_t_np *handle`
: The function handle

`grpc_session_attr_t_np *attr`
: The session attributes to be initialized

DESCRIPTION

The grpc_session_attr_initialize_with_object_handle_np() function initializes session attributes.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_OBJECT_HANDLE_NP`
: Object handle specified by handle is invalid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_session_attr_destruct_np` - Destructs session attributes.

SYNOPSIS

```
grpc_error_t grpc_session_attr_destruct_np(grpc_session_attr_t_np *attr)
```

ARGUMENTS

`grpc_session_attr_t_np *attr`
: The session attributes to be destructed

DESCRIPTION

The grpc_session_attr_destruct_np() function destructs session attributes.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_session_attr_get_np` - Gets handle attributes.

SYNOPSIS

```
grpc_error_t grpc_session_attr_get_np(grpc_session_attr_t_np *attr, grpc_session_attr_name_t_np name, void
**value)
```

ARGUMENTS

`grpc_session_attr_t_np *attr`
: The session attributes

`grpc_session_attr_name_t_np name`
: The names of the attributes to be get

`void *value`
: The values to be get

DESCRIPTION

The grpc_session_attr_get_np() function returns the values of session attributes.

See the manual of grpc_session_attr_set_np() for details of grpc_session_attr_name_t_np.

Memory area allocated for the attribute value should be released by calling grpc_session_attr_release_np() after the value was referred.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/08/04 02:32:49 $

NAME

`grpc_session_attr_set_np` - Sets session attributes.

SYNOPSIS

```
grpc_error_t grpc_session_attr_set_np(grpc_session_attr_t_np *attr, grpc_session_attr_name_t_np name, void
*value)
```

ARGUMENTS

`grpc_session_attr_t_np *attr`
: The session attributes

`grpc_session_attr_name_t_np name`
: The names of the attributes to be set

`void *value`
: The values to be set

```
typedef enum grpc_session_attr_name_e_np {
    GRPC_SESSION_ATTR_WAIT_ARG_TRANSFER,  /* grpc_argument_transfer_t_np */
    GRPC_SESSION_ATTR_SESSION_TIMEOUT,    /* int */
} grpc_session_attr_name_t_np;
```

- `GRPC_SESSION_ATTR_WAIT_ARG_TRANSFER`

  This flag specifies whether or not to wait for the transfer of arguments in an asynchronous RPC.

  The default is to wait for the transfer.

  The value set up with this attribute is shown below.

  - `GRPC_ARGUMENT_TRANSFER_WAIT`

    It waits for the end of transfer argument.

  - `GRPC_ARGUMENT_TRANSFER_NOWAIT`

    It does not wait for the end of transfer argument.

  - `GRPC_ARGUMENT_TRANSFER_COPY`

    The copy of an argument is made.

- `GRPC_SESSION_ATTR_SESSION_TIMEOUT`

  This specifies the RPC execution timeout time. The unit is in second.

  The details of this attribute is described in section 4.3.7, The client configuration file FUNCTION_INFO section.

DESCRIPTION

The grpc_session_attr_set_np() function sets the values of session attributes.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 08:13:35 $

NAME

`grpc_session_attr_release_np` - frees memory for the session attribute value.

SYNOPSIS

```
grpc_error_t grpc_session_attr_release_np(void *value)
```

ARGUMENTS

`void *value`
: Pointer to the value obtained by grpc_session_attr_get_np()

DESCRIPTION

The grpc_session_attr_release_np() frees memory for the value obtained by grpc_session_attr_get_np().

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, an error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/08/04 02:32:49 $

NAME

`grpc_arg_stack_new` - Prepares a stack for the arguments.

SYNOPSIS

```
grpc_arg_stack_t *grpc_arg_stack_new(int args)
```

ARGUMENTS

`int args`
    Number of arguments

DESCRIPTION

The grpc_arg_stack_new() function prepares a stack for passing to grpc_call_arg_stack() and grpc_call_arg_stack_async().

This function is MT-safe.

RETURN VALUE

If successful, the stack pointer is returned. If failed, NULL is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
    The grpc_initialize() function has not been executed.
`GRPC_OTHER_ERROR_CODE`
    Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_arg_stack_destruct` - Destructs a stack

SYNOPSIS

```
int grpc_arg_stack_destruct(grpc_arg_stack_t *stack)
```

ARGUMENTS

`grpc_arg_stack *stack`
  The stack

DESCRIPTION

The grpc_arg_stack_destruct() function destructs a stack.

This function is MT-safe.

RETURN VALUE

Returns 0 if successful. Returns -1 in the event of an error.

ERRORS

`GRPC_NOT_INITIALIZED`
  The grpc_initialize() function has not been executed.

`GRPC_OTHER_ERROR_CODE`
  Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_arg_stack_push_arg` - Pushes an argument onto the stack.

SYNOPSIS

```
int grpc_arg_stack_push_arg(grpc_arg_stack_t *stack, void *arg)
```

ARGUMENTS

`grpc_arg_stack_t *stack`
: The stack

`void *arg`
: Argument pointer

DESCRIPTION

The grpc_arg_stack_push_arg() function pushes the specified argument onto the stack.

This function is MT-safe.

RETURN VALUE

If successful, 0 is returned. In the case of an error, -1 is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: The grpc_initialize() function has not been executed.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_arg_stack_pop_arg` - Gets an argument from the stack.

SYNOPSIS

```
void *grpc_arg_stack_pop_arg(grpc_arg_stack_t *stack)
```

ARGUMENTS

`grpc_arg_stack_t *stack`
    The stack

DESCRIPTION

The grpc_arg_stack_pop_arg() function gets one argument pointer stored in the stack and returns it.

This function is MT-safe.

RETURN VALUE

If successful, a pointer to an argument is returned. If failed, NULL is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
    The grpc_initialize() function has not been executed.
`GRPC_OTHER_ERROR_CODE`
    Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_wait` - Waits for a session to end.

SYNOPSIS

```
grpc_error_t grpc_wait(grpc_sessionid_t session_id)
```

ARGUMENTS

`grpc_sessionid_t session_id`
: The session ID

DESCRIPTION

The grpc_wait() function waits for the specified session to end.

This function is MT-safe unless multiple wait functions wait the same sessions simultaneously.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_SESSION_ID`
: Session ID is not valid.

`GRPC_COMMUNICATION_FAILED`
: Communication with the server failed somehow.

`GRPC_SESSION_FAILED`
: The specified session failed.

`GRPC_TIMEOUT_NP`
: Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2007/07/27 02:43:36 $

NAME

`grpc_wait_all` - Waits until all sessions have ended.

SYNOPSIS

```
grpc_error_t grpc_wait_all()
```

ARGUMENTS

None

DESCRIPTION

The grpc_wait_all() function waits for all of the executing sessions to end. When no executing sessions exist, the grpc_wait_all() returns GRPC_NOERROR.

This function is MT-unsafe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SESSION_FAILED`
: The specified session failed.

`GRPC_TIMEOUT_NP`
: Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2007/07/27 02:43:36 $

NAME

`grpc_wait_any` - Waits for any session to end.

SYNOPSIS

```
grpc_error_t grpc_wait_any(grpc_sessionid_t *idPtr)
```

ARGUMENTS

`grpc_sessionid_t *idPtr`
: Pointer to an area for returning the session ID of the session that has completed execution

DESCRIPTION

The grpc_wait_any() function waits for any one of the currently executing sessions to end. When no executing sessions exist, the grpc_wait_any() function returns GRPC_NOERROR and sets GRPC_SESSIONID_VOID to the area pointed by idPtr.

This function is MT-unsafe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_SESSION_FAILED`
: The specified session failed.

`GRPC_TIMEOUT_NP`
: Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2007/07/27 02:43:36 $

NAME

`grpc_wait_and` - Waits for multiple sessions to end.

SYNOPSIS

```
grpc_error_t grpc_wait_and(grpc_sessionid_t *idArray, size_t length)
```

ARGUMENTS

`grpc_sessionid_t *idArray`
    Pointer to the session IDs

`size_t length`
    The number of session IDs stored in sessions

DESCRIPTION

The grpc_wait_and() function waits for all of the sessions specified by sessions to end.

This function is MT-safe unless multiple wait functions wait the same sessions simultaneously.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
    GRPC client is not initialized yet.

`GRPC_INVALID_SESSION_ID`
    Session ID is not valid.

`GRPC_SESSION_FAILED`
    The specified session failed.

`GRPC_TIMEOUT_NP`
    Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
    Internal error detected.

---

last update : $Date: 2007/07/27 02:43:36 $

NAME

`grpc_wait_or` - Waits for any session to end.

SYNOPSIS

```
grpc_error_t grpc_wait_or( grpc_sessionid_t *idArray, size_t length, grpc_sessionid_t *idPtr)
```

ARGUMENTS

`grpc_sessionid_t *idArray`
: A pointer to the session IDs

`size_t length`
: The number of session IDs stored in sessions

`grpc_sessionid_t *idPtr`
: Pointer to an area for returning the session ID of the session that has completed execution

DESCRIPTION

The grpc_wait_or() function sessions waits for any one of the specified sessions to end.

This function is MT-safe unless multiple wait functions wait the same sessions simultaneously.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_SESSION_ID`
: Session ID is not valid.

`GRPC_SESSION_FAILED`
: The specified session failed.

`GRPC_TIMEOUT_NP`
: Timeout occurred in session.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2007/07/27 02:43:36 $

NAME

`grpc_cancel` - Cancels a session.

SYNOPSIS

```
grpc_error_t grpc_cancel(grpc_sessionid_t session_id)
```

ARGUMENTS

`grpc_sessionid_t session_id`
: The session ID

DESCRIPTION

The grpc_cancel() function cancels the current session.

grpc_cancel() is a non-blocking function. It does not wait for the completion of the cancellation.

A cancelled session should be taken care by a wait function such as grpc_wait() so that the allocated resources for the session can be released.

Wait functions such as grpc_wait() return GRPC_SESSION_FAILED if those functions detect the cancelled session. grpc_get_error(sessionID) returns GRPC_CANCELED_NP if the cancel was successfully completed. Otherwise, it returns an error.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_SESSION_ID`
: Session ID is not valid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_cancel_all` - Cancels all sessions.

SYNOPSIS

```
grpc_error_t grpc_cancel_all()
```

ARGUMENTS

`None`

DESCRIPTION

The grpc_cancel_all() function cancels all of the executing sessions.

grpc_cancel_all() is a non-blocking function. It does not wait for the completion of the cancellation.

A cancelled session should be taken care by a wait function such as grpc_wait() so that the allocated resources for the session can be released.

Wait functions such as grpc_wait() return GRPC_SESSION_FAILED if those functions detect the cancelled session. grpc_get_error(sessionID) returns GRPC_CANCELED_NP if the cancel was successfully completed. Otherwise, it returns an error.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_probe` - Checks the session execution status.

SYNOPSIS

```
grpc_error_t grpc_probe(grpc_sessionid_t session_id)
```

ARGUMENTS

`grpc_sessionid_t session_id`
: The session ID

DESCRIPTION

The grpc_probe() function returns the execution status of the session specified by the session ID.

This function is MT-safe.

RETURN VALUE

If the session is completed, GRPC_NO_ERROR is returned. Otherwise, GRPC_NOT_COMPLETED is returned. If
the grpc_probe itself failed, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_SESSION_ID`
: Session ID is not valid.

`GRPC_NOT_COMPLETED`
: Call has not completed.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_probe_or` - Checks the session execution status.

SYNOPSIS

```
grpc_error_t grpc_probe_or( grpc_sessionid_t *sessions, size_t length, grpc_sessionid_t *id)
```

ARGUMENTS

`grpc_sessionid_t *sessions`
: A pointer to the session IDs

`size_t length`
: The number of session IDs stored in sessions

`grpc_sessionid_t *id`
: Pointer to an area for returning the session ID of the session that has completed execution

DESCRIPTION

The grpc_probe_or() function returns the execution status of the sessions specified by session IDs.
If no sessions have been completed, the function returns an error.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_SESSION_ID`
: Session ID is not valid.

`GRPC_OTHER_ERROR_CODE`
: Internal error is detected.

`GRPC_NONE_COMPLETED`
: No sessions have been completed.

---

last update : $Date: 2005/08/03 09:23:42 $

NAME

`grpc_session_info_get_np` - Get session information.

SYNOPSIS

`grpc_error_t grpc_session_info_get_np(grpc_sessionid_t session_id, grpc_session_info_t_np **info, int *status)`

ARGUMENTS

`grpc_sessionid_t session_id`
: The session ID

`grpc_session_info_t_np **info`
: The session information

`int *status`
: The session status

DESCRIPTION

The grpc_session_info_get_np() function returns information on the specified session.
If NULL is specified in info, only the session status is returned as the return value.
The storage of session information was allocated in this function. Release the session information by grpc_session_info_release_np() when it becomes unnecessary.
grpc_session_info_t_np is defined in $NG_DIR/include/grpc.h.
The time of compression is included in members of grpc_session_info_t_np listed following.

```
/**
 * Session Infomation
 */
/* Measured by the remote method */
typedef struct grpc_exec_info_executable_s_np {
    int callbackNtimesCalled;

    /* The time concerning argument transmission */
    struct timeval transferArgumentToRemoteRealTime;
    struct timeval transferArgumentToRemoteCpuTime;

    /* The time concerning transfer file from client to remote */
    struct timeval transferFileToRemoteRealTime;
    struct timeval transferFileToRemoteCpuTime;

    /* The time of Calculation time of executable */
    struct timeval calculationRealTime;
    struct timeval calculationCpuTime;

    /* The time concerning transmitting a result */
    struct timeval transferResultToClientRealTime;
    struct timeval transferResultToClientCpuTime;

    /* The time concerning transfer file from client to remote */
    struct timeval transferFileToClientRealTime;
    struct timeval transferFileToClientCpuTime;

    /* The time concerning argument transmission of callback */
    struct timeval callbackTransferArgumentToClientRealTime;
    struct timeval callbackTransferArgumentToClientCpuTime;

    /* The time concerning callback */
    struct timeval callbackCalculationRealTime;
    struct timeval callbackCalculationCpuTime;

    /* The time concerning transmitting a result of callback */
    struct timeval callbackTransferResultToRemoteRealTime;
    struct timeval callbackTransferResultToRemoteCpuTime;
} grpc_exec_info_executable_t_np;
```

```
    /* Measured by the client */
    typedef struct grpc_exec_info_client_s_np {
        int callbackNtimesCalled;

        /* The time concerning request remote machine information */
        struct timeval remoteMachineInfoRequestRealTime;
        struct timeval remoteMachineInfoRequestCpuTime;

        /* The time concerning request remote class information */
        struct timeval remoteClassInfoRequestRealTime;
        struct timeval remoteClassInfoRequestCpuTime;

        /* The time concerning invoke GRAM */
        struct timeval gramInvokeRealTime;
        struct timeval gramInvokeCpuTime;

        /* The time concerning argument transmission */
        struct timeval transferArgumentToRemoteRealTime;
        struct timeval transferArgumentToRemoteCpuTime;

        /* The Calculation time of client */
        struct timeval calculationRealTime;
        struct timeval calculationCpuTime;

        /* The time concerning transmitting a result */
        struct timeval transferResultToClientRealTime;
        struct timeval transferResultToClientCpuTime;

        /* The time concerning argument transmission of callback */
        struct timeval callbackTransferArgumentToClientRealTime;
        struct timeval callbackTransferArgumentToClientCpuTime;

        /* The time concerning calculation of callback */
        struct timeval callbackCalculationRealTime;
        struct timeval callbackCalculationCpuTime;

        /* The time concerning transmitting a result of callback */
        struct timeval callbackTransferResultToRemoteRealTime;
        struct timeval callbackTransferResultToRemoteCpuTime;

    } grpc_exec_info_client_t_np;

    /* Compression Information */
    typedef struct grpc_compression_info_s_np {
        int         valid;  /* data below valid? 0:invalid, 1:valid */

        /* Number of bytes of data before compression */
        size_t      originalNbytes;

        /* Number of bytes of data after compression */
        size_t      compressionNbytes;

        /* Lapsed time at the time of compression */
        struct timeval compressionRealTime;
        struct timeval compressionCpuTime;

        /* Lapsed time at the time of decompression */
        struct timeval decompressionRealTime;
        struct timeval decompressionCpuTime;
    } grpc_compression_info_t_np;

    /* Session Information */
    typedef struct grpc_session_info_s_np {
        grpc_exec_info_executable_t_np gei_measureExecutable;
        grpc_exec_info_client_t_np     gei_measureClient;

        struct {
            /* Number of elements as toRemote and toClient */
            int nElements;
            grpc_compression_info_t_np *toRemote;
            grpc_compression_info_t_np *toClient;
        } gei_compressionInformation;
    } grpc_session_info_t_np;
```

Refer to the following for return status.

GRPC_SESSION_ARG_IS_NOT_TRANSMITTED
Transmission of the arguments to the stub has not been completed.

GRPC_SESSION_EXECUTING
The session is in progress.

GRPC_SESSION_DOWN
The session is not being executed.

GRPC_SESSION_DONE
The session has ended.

GRPC_SESSION_UNKNOWN_STATUS
API was failed.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
    GRPC client is not initialized yet.
`GRPC_INVALID_SESSION_ID`
    Session ID is not valid.
`GRPC_OTHER_ERROR_CODE`
    Internal error detected.

---

last update : $Date: 2005/07/11 08:13:35 $

NAME

`grpc_session_info_release_np` - Release session information.

SYNOPSIS

```
grpc_error_t grpc_session_info_release_np( grpc_session_info_t_np *info)
```

ARGUMENTS

`grpc_session_info_t_np *info`
    The session information

DESCRIPTION

The grpc_session_info_release_np() function release the session information that allocated by grpc_session_info_get_np().
Returned immediately if NULL is specified in info.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

---

last update : $Date: 2005/07/11 08:13:35 $

NAME

`grpc_session_info_threshold_np` - Sets the number of session information units to be saved.

SYNOPSIS

```
grpc_error_t grpc_session_info_threshold_np(int threshold)
```

ARGUMENTS

`int threshold`
: The number of session information units to be saved

DESCRIPTION

The grpc_session_info_threshold_np() function sets the number of session information units to be saved. If a negative value is specified in threshold, discarding is not done automatically.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_session_info_remove_np` - grpc_session_info_remove_np - Discards session information.

SYNOPSIS

```
grpc_error_t grpc_session_info_remove_np(grpc_sessionid_t session_id)
```

ARGUMENTS

`grpc_sessionid_t session_id`
: The session ID

DESCRIPTION

The grpc_session_info_remove_np() function discards the information on the specified session. If the value specified in session_id is -1, all of the session information is discarded.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_INVALID_SESSION_ID`
: Session ID is not valid.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_get_error` - Returns the error code generated by the session.

SYNOPSIS

```
grpc_error_t grpc_get_error(grpc_sessionid_t session_id)
```

ARGUMENTS

`grpc_sessionid_t session_id`
    The session ID

DESCRIPTION

The grpc_get_error() function returns the error code that was generated in the session specified by session ID.

This function is MT-safe.

RETURN VALUE

If successful, the error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
    GRPC client is not initialized yet.
`GRPC_INVALID_SESSION_ID`
    Session ID is not valid.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_error_string` - Returns the error message for the error code.

SYNOPSIS

```
char *grpc_error_string(grpc_error_t error_code)
```

ARGUMENTS

`grpc_error_t error_code`
: The error code

DESCRIPTION

The grpc_error_string() function returns the error message that corresponds to the error code.

This function is MT-safe.

RETURN VALUE

The error message that corresponds to the specified error code is returned.
If a nonexistent error code is specified, "GRPC_UNKNOWN_ERROR_CODE" is returned.

ERRORS

`None`

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_perror_np` - Displays the last error that occurred.

SYNOPSIS

```
grpc_error_t grpc_perror_np(char *str)
```

ARGUMENTS

`char *str`
: The character string to be displayed

DESCRIPTION

The grpc_perror_np() function displays the last error that occurred in the standard error output.

This function is MT-safe.

RETURN VALUE

GRPC_NO_ERROR is returned.

ERRORS

`None`

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_get_failed_sessionid` - Get session ID that failed for calls.

SYNOPSIS

```
grpc_error_t grpc_get_failed_sessionid(grpc_sessionid_t *idPtr)
```

ARGUMENTS

`grpc_sessionid_t *idPtr`
: The session ID

DESCRIPTION

The grpc_get_failed_sessionid() function returns the session ID associated with the most recent GRPC_SESSION_FAILED error. This provides additional error information on a specific session ID that failed for calls that deal with sets of session IDs, either implicitly, such as grpc_wait_all(), or explicitly, such as grpc_wait_and().

When there are more than two failed sessions, this function will return the session ID one by one. To make sure that all the failed sessions are handled, users have to call this function repeatedly until it returns GRPC_SESSIONID_VOID.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/08/04 02:40:32 $

NAME

`grpc_last_error_get_np` - Returns the error code of the last error to occur.

SYNOPSIS

```
grpc_error_t grpc_last_error_get_np()
```

ARGUMENTS

```
None
```

DESCRIPTION

The grpc_last_error_get_np() function returns the error code of the last error to occur.

This function is MT-safe.

RETURN VALUE

The error code of the last error to occur

ERRORS

```
None
```

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_signal_handler_set_np` - Set the signal handler.

SYNOPSIS

```
grpc_error_t grpc_signal_handler_set_np(int sig_num, void (*sig_handler)(int))
```

ARGUMENTS

`int sig_num`
: The signal whose handler is modified.

`void (*sig_handler)(int))`
: The address of a signal handler.

DESCRIPTION

The grpc_signal_handler_set_np() function modifies signal dispositions for Ninf-G Client.

Procedures for signal handling differs according to environments and a signal to be processed.

- pthread flavors of GT 4.0.0 or later

  Ninf-G uses signal handling API provided by GT4, but the API does not support SIGKILL, SIGSEGV, SIGABRT, SIGBUS, SIGFPE, SIGILL, SIGIOT, SIGPIPE, SIGEMT, SIGSYS, SIGTRAP, SIGSTOP, SIGCONT and SIGWAITING.

  If sig_num is supported by GT's signal handling API, sig_handler is called by a signal handling thread.

  if sig_num is not supported by GT's signal handling API, sig_handler is called as a signal handler registered by sigaction().

- pthread flavors of earlier than GT 4.0.0

  Ninf-G has signal handling thread which supports the same signals as GT4 signal handling API.

  If sig_num is supported by a Ninf-G signal handling thread, sig_handler is called by the signal handling thread. Otherwise, sig_handler is called as a signal handler registered by sigaction().

  Note: For pthread flavor on MacOS X, Ninf-G Client processes SIGSTP as well.

- nonthread flavors for all versions of GT.

  sig_handler is called as a signal handler registered by sigaction().

It is unsafe to call some system calls from the signal handler registered by sigaction(). A list of safe system calls is available on the following web page and IEEE Std 1003.1(POSIX).

http://www.opengroup.org/onlinepubs/007908799/xsh/sigaction.html

This function is a new function in Ninf-G version 4.0.0.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2006/09/29 07:31:53 $

NAME

`grpc_is_canceled_np` - Checks whether the Ninf-G Client has requested cancellation.

SYNOPSIS

```
#include <grpc_executable.h>

int grpc_is_canceled_np()
```

ARGUMENTS

`grpc_error_t *error`
: If successful, GRPC_NO_ERROR is returned to *error. In the case of an error, Error code is returned to *error.

DESCRIPTION

The grpc_is_canceled_np() function checks whether the Ninf-G Client has requested cancellation or not.

Note: grpc_is_canceled_np() is defined as Ninf-G Executable API, which is only used by the server side program.

RETURN VALUE

If the Ninf-G Client requested the session cancel by grpc_cancel(), this function returns 1. Otherwise, this function returns 0.

ERRORS

None

---

last update : $Date: 2007/07/10 05:19:42 $

NAME

`grpc_get_info_np` - Returns session information.

SYNOPSIS

`grpc_error_t grpc_get_info_np(grpc_sessionid_t session_id, grpc_exec_info_t_np *info, int *status)`

ARGUMENTS

`grpc_sessionid_t session_id`
: The session ID

`grpc_exec_info_t_np *info`
: The session information

`int *status`
: The session status

DESCRIPTION

Not recommend. Use grpc_session_info_get_np() instead.
The grpc_get_info_np() function returns information on the specified session.
If NULL is specified in info, only the session status is returned as the return value.
grpc_exec_info_t_np is defined in $NG_DIR/include/grpc.h.
The time of compression is included in members of grpc_exec_info_t_np listed following.

- transferArgumentRealTime
- transferArgumentCPUTime
- transferResultRealTime
- transferResultCPUTime
- callbackTransferArgumentRealTime
- callbackTransferArgumentCPUTime
- callbackTransferResultRealTime
- callbackTransferResultCPUTime

```
typedef struct grpc_exec_info_s_np {
    struct {
        /* Measured by the remote method */

        /* Real time of the time concerning argument transmission */
        struct timeval transferArgumentRealTime;
        /* CPU time of the time concerning argument transmission */
        struct timeval transferArgumentCpuTime;
        /* Real time of Calculation time of executable */
        struct timeval calculationRealTime;
        /* CPU time of Calculation time of executable */
        struct timeval calculationCpuTime;
        /* Real time of the time concerning transmitting a result */
        struct timeval transferResultRealTime;
        /* CPU time of the time concerning transmitting a result */
        struct timeval transferResultCpuTime;

        /* Real time of the time concerning argument transmission of callback */
        struct timeval callbackTransferArgumentRealTime;
        /* CPU time of the time concerning argument transmission of callback */
        struct timeval callbackTransferArgumentCpuTime;
        /* Real time of time concerning callback */
        struct timeval callbackCalculationRealTime;
        /* CPU time of time concerning callback */
        struct timeval callbackCalculationCpuTime;
        /* Real time of the time concerning transmitting a result of callback */
        struct timeval callbackTransferResultRealTime;
        /* CPU time of the time concerning transmitting a result of callback */
        struct timeval callbackTransferResultCpuTime;
    } gei_measureExecutable;
    struct {
        /* Measured by the client */
        /* Real time of the time concerning request remote machine information */
        struct timeval remoteMachineInfoRequestRealTime;
        /* CPU time of the time concerning request remote machine information */
        struct timeval remoteMachineInfoRequestCpuTime;
        /* Real time of the time concerning request remote class information */
        struct timeval remoteClassInfoRequestRealTime;
        /* CPU time of the time concerning request remote class information */
        struct timeval remoteClassInfoRequestCpuTime;
        /* Real time of the time concerning invoke GRAM */
        struct timeval gramInvokeRealTime;
        /* CPU time of the time concerning invoke GRAM */
        struct timeval gramInvokeCpuTime;

        /* Real time of the time concerning argument transmission */
        struct timeval transferArgumentRealTime;
        /* CPU time of the time concerning argument transmission */
        struct timeval transferArgumentCpuTime;
        /* Real time of Calculation time of client */
        struct timeval calculationRealTime;
        /* CPU time of Calculation time of client */
        struct timeval calculationCpuTime;
        /* Real time of the time concerning transmitting a result */
        struct timeval transferResultRealTime;
        /* CPU time of the time concerning transmitting a result */
        struct timeval transferResultCpuTime;

        /* Real time of the time concerning argument transmission of callback */
        struct timeval callbackTransferArgumentRealTime;
        /* CPU time of the time concerning argument transmission of callback */
        struct timeval callbackTransferArgumentCpuTime;
        /* Real time of time concerning calculation of callback */
        struct timeval callbackCalculationRealTime;
        /* CPU time of time concerning calculation of callback */
        struct timeval callbackCalculationCpuTime;
        /* Real time of the time concerning transmitting a result of callback */
        struct timeval callbackTransferResultRealTime;
        /* CPU time of the time concerning transmitting a result of callback */
        struct timeval callbackTransferResultCpuTime;
    } gei_measureClient;
} grpc_exec_info_t_np;
```

Refer to the following for return status.

GRPC_SESSION_ARG_IS_NOT_TRANSMITTED
Transmission of the arguments to the stub has not been completed.

GRPC_SESSION_EXECUTING
The session is in progress.

GRPC_SESSION_DOWN
The session is not being executed.

GRPC_SESSION_DONE
The session has ended.

GRPC_SESSION_UNKNOWN_STATUS
API was failed.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

GRPC_NOT_INITIALIZED

GRPC client is not initialized yet.

`GRPC_INVALID_SESSION_ID`

Session ID is not valid.

`GRPC_OTHER_ERROR_CODE`

Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

NAME

`grpc_get_last_info_np` - Returns the information on the last session that was run.

SYNOPSIS

```
grpc_error_t grpc_get_last_info_np(grpc_exec_info_t_np *info, int *status)
```

ARGUMENTS

`grpc_exec_info_t_np *info`
: The session information

`int *status`
: The session status

DESCRIPTION

Not recommend. Use grpc_session_info_get_np() instead.
The grpc_get_last_info_np() function returns the information on the last-executed session.
If NULL is specified in info, only the status of the session is returned as the return value.
See the manual of grpc_get_info_np() for details of info.

Refer to the following for return status.

GRPC_SESSION_ARG_IS_NOT_TRANSMITTED
Transmission of the arguments to the stub has not been completed.

GRPC_SESSION_EXECUTING
The session is in progress.

GRPC_SESSION_DOWN
The session is not being executed.

GRPC_SESSION_DONE
The session has ended.

GRPC_SESSION_UNKNOWN_STATUS
API was failed.

This function is MT-safe.

RETURN VALUE

If successful, GRPC_NO_ERROR is returned. In the case of an error, Error code is returned.

ERRORS

`GRPC_NOT_INITIALIZED`
: GRPC client is not initialized yet.

`GRPC_OTHER_ERROR_CODE`
: Internal error detected.

---

last update : $Date: 2005/07/11 07:11:24 $

8. Ninf-G Utility Command Reference Manual

This is the reference manual for the utility command provided by Ninf-G. (The text is in the format of a UNIX on-line manual.)

---

last update : $Date: 2005/11/18 08:19:14 $

NAME

`ng_cc` - The Ninf-G Client compiler

SYNOPSIS

```
ng_cc [compiler options]
```

DESCRIPTION

The ng_cc is a script that wraps the compiler and linker. It generates Ninf-G Client programs by compiling and linking application programs.

Options and arguments for the compiler and linker can be written on the ng_cc command line. Those are passed to the compiler and linker used in ng_cc .
For example, executing the following command will to generate Ninf-G Client(test_client) from the test_client.c application program using the default (C language) compiler and linker.

```
    % ng_cc -g -o test_client test_client.c
```

The default compiler and linker used by ng_cc is the C compiler and linker, cc. If the application program is written in C and the cc compiler and linker is used, executing the ng_cc command will create an Ninf-G Client.

A compiler and linker other than cc can be used by setting the NG_COMPILER and NG_LINKER environment variables to specify the compiler and linker to be used by ng_cc.

Note: Mixed utilization of off_t and other file size related data types may cause mismatch of data size. By default, ng_cc uses large file option such as _FILE_OFFSET_BITS=64 (on Linux) as its compile option. Thus, the size of off_t type compiled by ng_cc may differ from the size of off_t type compiled by non ng_cc command.

ERRORS

If the compiling of the application program fails, the compiler error message is output.

---

last update : $Date: 2008/03/12 09:07:12 $

NAME

`ng_delete_functions` - Deletes function information

SYNOPSIS

```
ng_delete_functions [-f] Module[/Entry] ...
```

DESCRIPTION

The ng_delete_functions command deletes the specified function information from the MDS, making that information unavailable from the MDS.

Specifying a list of "module name" or "module name/entry name" in the argument will result in the corresponding information being deleted from the MDS.
If "module name/entry name" is specified, the corresponding information will be deleted. If "module name" only is specified, all of the function information that pertains to that module will be deleted.

When the information is deleted, a confirmation is performed for the deletion of each entry.
If the -f option is specified, the deletion confirmations are not performed.

When the deletion operation is successful, a list of the deleted entry names is output.

OPTIONS

-f
: The delete confirmation is not performed.

ERRORS

If an attempt is made to delete an entry for which there is no delete permission, a message is output to inform of the lack of permission.

---

last update : $Date: 2005/07/11 07:13:44 $

NAME

`ng_dump_functions` - outputs function information

SYNOPSIS

```
ng_dump_functions [Module[/Entry] ...]
```

DESCRIPTION

The ng_dump_functions command outputs the function information provided by the MDS of the host on which the command was executed. If a "module name" or "module name/entry name" is specified in the argument, the corresponding function information will be output; if nothing is specified, all of the function information provided by that host will be output.

The following information is output.
- The owner of the function
- The module name
- The entry name

---

last update : $Date: 2005/07/11 07:13:44 $

NAME

`ng_gen` - ng_gen - Ninf-G stub generator

SYNOPSIS

```
ng_gen [-d] [-g] [-h] inputfile
```

DESCRIPTION

The ng_gen command interprets the Ninf-G IDL file and generates what is needed to create an Ninf-G Executable . The Ninf-G IDL is specified by the argument.

The ng_gen command executes cpp to process the Ninf-G IDL file.
If unsupported option strings(beginning with '-') are passed to ng_gen command, it handles them as options to cpp.

The ng_gen command generates the following items.

- Ninf-G stub
- Makefile for the Ninf-G Executable
- LDIF file
- Local LDIF file

OPTIONS

-d
: Dump information about function.

-g
: Produce makefile in gnu make style.

-h
: Print usage.

--no-cpp
: Do not use cpp.

--with-cpp=cpp_command
: Use the specified cpp instead of the default cpp.

ERRORS

If the input Ninf-G IDL is incorrect, an error message is output and the process ends abnormally.

---

last update : $Date: 2006/02/17 04:21:01 $

NAME

`ng_version` - print Ninf-G version

SYNOPSIS

```
ng_version [-v]
```

DESCRIPTION

The ng_version prints version of Ninf-G.

OPTIONS

-v
: Prints configure options performed on compiled time.

---

last update : $Date: 2005/07/11 07:13:44 $

Invoke Server Developer's Manual

This document describes how to develop a Ninf-G Invoke Server.

---

1. Introduction

A Ninf-G Client invokes a Ninf-G Executable on the server machine when a function requiring initialization of function/object handles, such as grpc_function_handle_init(), is called. Ninf-G, Version 2, implements the remote process invocation using the Globus Toolkit's Pre-WS GRAM feature. Implemented using the Globus API, the invocation mechanism has been embedded in Ninf-G. In order to utilize other systems, such as WS GRAM, UNICORE, or Condor for remote process invocation, Ninf-G, Version 4, implements the invocation mechanism as a separate module called "Invoke Server." This design enables users and developers to implement and add a new Invoke Server that can utilize any job invocation mechanism.

Ninf-G Version 4.2.0 includes the following Invoke Servers:

- Invoke Server for WS GRAM, implemented in Python (GT4py)
- Invoke Server for SSH, implemented in C (SSH)
- Invoke Server for Condor, implemented in Java (Condor)
- Invoke Server for Pre-WS GRAM, implemented in C (GT2c)
- Invoke Server for WS GRAM, implemented in Java (GT4java)
- Invoke Server for UNICORE, implemented in Java (UNICORE)
- Invoke Server for NAREGI Super Scheduler, implemented in Java (NAREGISS)

1.1 Overview of a typical client application

Here is a typical flow of a Ninf-G Client application:

- (1) grpc_initialize()

  Initializes data structures used by the Ninf-G Client.

- (2) grpc_function_handle_init()

  Creates a function/object handle which requests remote process invocation. The request will be processed and a Ninf-G Executable will be created on the server machine. When the Ninf-G Executable is created, it connects to the Ninf-G Client to establish a TCP connection between the Ninf-G Executable and the Ninf-G Client.

- (3) grpc_call() or grpc_call_async()/grpc_wait_any()

  Calls the remote function, i.e. (3.1) the Ninf-G Client sends arguments to the Ninf-G Executable, (3.2) the Ninf-G Executable performs some form of computation, and (3.3) the Ninf-G Executable sends the results to the Ninf-G Client.

- (4) grpc_function_handle_destruct()

  Requests the Ninf-G Executable to terminate its process. If an error occurs during the termination, the Ninf-G Client requests the Invoke Server to kill the Ninf-G Executable.

- (5) grpc_finalize()

  Frees the data structures used by the Ninf-G Client.

Invoke Server is required to implement initialization and finalization of the function/object handles which are described in steps (2) and (4).

1.2 Requirements for underlying middleware

The only requirement for underlying middleware is that the middleware must be capable of remote process invocation. Examples of such middleware include the Globus Toolkit Pre-WS GRAM, Globus Toolkit WS GRAM, Unicore, Condor, and SSH.

1.3 Implementation overview

Invoke Server is an adapter for the underlying middleware and it handles requests from a Ninf-G Client. Invoke Server analyzes and processes the request sent from the Ninf-G Client and replies to the Ninf-G Client. For example, if Invoke Server receives a `JOB_CREATE` request from the Ninf-G Client, Invoke Server creates a Job ID, returns the Job ID to the Ninf-G Client, and invokes the job processes called for in the request.

Invoke Server can be implemented using any language. The details of the protocol existing between the Ninf-G Client and Invoke Server are described in Section 2.

1.4 Execution flow

This section describes a sample RPC flow to a server called serverA via the Invoke Server, `IS_SAMPLE`.

- (Prerequisite)
- (1) A client configuration file that describes that Invoke Server `IS_SAMPLE` is used for RPC to serverA must be prepared.
- (grpc_function_handle_init())
- (2) The Ninf-G Client requests Invoke Server `IS_SAMPLE` to create a function/object handle.
- (3) The first time `IS_SAMPLE` is required to create a function/object handle, the `IS_SAMPLE` process is spawned by the Ninf-G Client on the same machine. `${NG_DIR}/bin/ng_invoke_server.IS_SAMPLE` is a command for spawning an `IS_SAMPLE` process.
- (4) The Ninf-G Client and `IS_SAMPLE` communicate using three pipes (stdin, stdout, and stderr).

- (5) When grpc_function_handle_init() is called, the Ninf-G Client sends `JOB_CREATE` request to `IS_SAMPLE`, followed by the required information (e.g., the hostname and port number of the remote server), and `JOB_CREATE_END`.
- (6) When `IS_SAMPLE` receives `JOB_CREATE` request, `IS_SAMPLE` returns "`S`" to the Ninf-G Client, which indicates that the request has been received by the Invoke Server.
- (7) `IS_SAMPLE` generates a new Job ID that corresponds to the Request ID that was transferred with the `JOB_CREATE` request, and notifies the Job ID to the Ninf-G Client. Then, `IS_SAMPLE` invokes the remote processes (Ninf-G Executable) on serverA using its underlying middleware.
- (8) The Ninf-G Client waits for the reply from `IS_SAMPLE`, and notify of Job ID. When the Ninf-G Client receives the reply and Job ID, it resumes the execution without waiting for actual job invocation on serverA.
- (grpc_call())
- (9) When the Ninf-G Executable is invoked on serverA, it connects to the Ninf-G Client using Globus IO. The connection is used for communication (e.g., argument transfers from the Ninf-G Client to the Ninf-G Executable) between the Ninf-G Client and the Ninf-G Executable. `IS_SAMPLE` does nothing for grpc_call(). If the underlying middleware for `IS_SAMPLE` returns an error on remote process invocation, `IS_SAMPLE` must notify the Ninf-G Client that the job invocation has failed.
- (grpc_function_handle_destruct())
- (10) When grpc_function_handle_destruct() is called, the Ninf-G Client requests the Ninf-G Executable to exit the process. This communication is carried out between the Ninf-G Client and the Ninf-G Executable. The Ninf-G Client does not wait for the Ninf-G Executables to be terminated.
- (11) When the Ninf-G Executable exits the process, the job status managed by `IS_SAMPLE` should be changed to `DONE`, and `IS_SAMPLE` notifies the Ninf-G Client of the change in job status to `DONE`.
- (12) The Ninf-G Client sends a `JOB_DESTROY` request to `IS_SAMPLE`.
- (13) `IS_SAMPLE` returns "`S`" to the Ninf-G Client when it receives the `JOB_DESTROY` request.
- (14) `IS_SAMPLE` returns `DONE` to the Ninf-G Client if the state of the corresponding job is `DONE`. Otherwise, `IS_SAMPLE` cancels the job and notifies the Ninf-G Client of the change in status to `DONE` when the cancellation is completed and the status of the job actually becomes `DONE`.
- (grpc_finalize())
- (15) When grpc_finalize() is called, the Ninf-G Client sends an `EXIT` request to `IS_SAMPLE`.
- (16) `IS_SAMPLE` returns "`S`" to the Ninf-G Client when it receives the `EXIT` request. The pipes between `IS_SAMPLE` and Ninf-G Client (stdin, stdout, stderr) are closed after it.
- (17) `IS_SAMPLE` cancels all jobs and wait the termination of all jobs, and exit.
- (18) When the Ninf-G Client receives an "`S`" from `IS_SAMPLE`, it continues its execution, and does not wait for the termination of all jobs.

The following figure illustrates the interaction between the Ninf-G Client, Invoke Server, and the Ninf-G Executable.

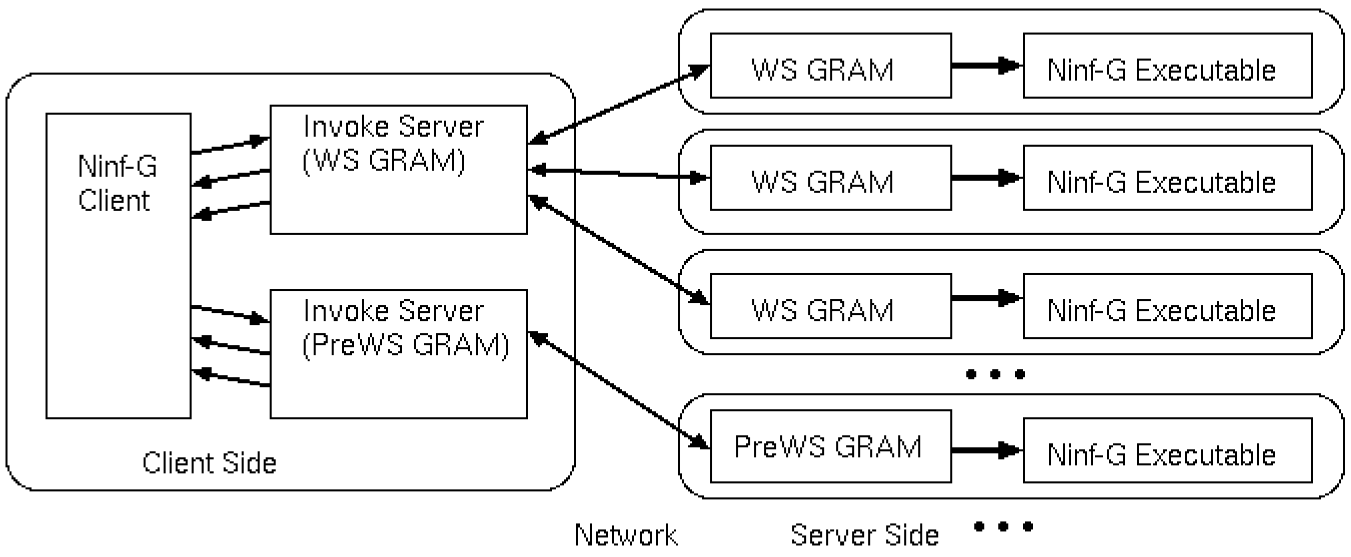

Figure 1: Interaction between the Ninf-G Client, Invoke Server and the Ninf-G Executable

2. Specifications of Invoke Server

This section describes a detailed overview of Invoke Server and the protocol existing between a Ninf-G Client and Invoke Server.

2.1 Detailed overview of Invoke Server

1. Invoke Server is invoked when a Ninf-G Client initializes a function/object handle on the remote server which Ninf-G Client is configured to use with Invoke Server.

2. The maximum number of jobs per Invoke Server is limited. If the number of jobs exceeds the limit, a new Invoke Server is invoked.

3. Invoke Server exits the process if it receives an `EXIT` request from the Ninf-G Client. This request is sent when the Ninf-G Client calls grpc_finalize(). Invoke Server also exits the process if it is managing the maximum number of jobs and all jobs are terminated.

4. The Ninf-G Client and Invoke Server communicate using three pipes, created by the Ninf-G Client when the Invoke Server is invoked.

5. Ninf-G Client does not wait for the termination of Invoke Server after the Ninf-G Client sends an `EXIT` request to Invoke Server.

6. If the Ninf-G Client exits abnormally, the pipes will be disconnected. When Invoke Server detects that the pipes have been disconnected, Invoke Server must cancel all jobs and exit the process.

7. Invoke Server is implemented as a Unix executable or script file which should be located in the `${NG_DIR}/bin` directory. It can be located in another directory if Invoke Server is supplied with an absolute path to the executable file.

8. The file names used with Invoke Server must follow the naming convention of "ng_invoke_server" + suffix, where the suffix corresponds to rules for the underlying middleware used for remote process invocation.

9. Log file for Invoke Server can be specified as an optional argument of the Invoke Server command.

   Example:

   ```
   -l [Log file name]
   ```

If this option is specified, Invoke Server outputs logs to the file specified by this argument. Otherwise, logs are not recorded.

2.2 Protocol between a Ninf-G Client and Invoke Server

2.2.1 Overview

A Ninf-G Client and Invoke Server exchange three types of messages, Request, Reply, and Notify. A Request message is sent from a Ninf-G Client to Invoke Server. Reply and Notify messages are sent from Invoke Server to the Ninf-G Client. The Ninf-G Client assumes that a Reply message must be returned from Invoke Server when the Ninf-G Client sends a Request message. A Notify message is used to send messages from Invoke Server to the Ninf-G Client asynchronously. Three different pipes are used for sending these three types of message.

Name	fd	direction		
Request	stdin	Ninf-G	---->	Invoke
Reply	stdout	Client	<----	Server
Notify	stderr		<----	

2.2.2 Protocol

All messages are sent as plain text. The Return code (<RET>) is `0x0d0a`. The Return code is a delimiter that determines the unit of messages. A Job ID is generated by Invoke Server.

2.2.2.1 Request

Four Request messages, `JOB_CREATE`, `JOB_STATUS`, `JOB_DESTROY`, and `EXIT` are supported.

1. `JOB_CREATE`
    - Format

      ```
      JOB_CREATE <Request ID><RET>
      hostname .....<RET>
      port .....<RET>
      ... (snip)
      JOB_CREATE_END<RET>
      ```

    - Explanation

      This request is used to create and invoke a new job. Required information for job invocation is described as a set of attributes that is transferred along with a `JOB_CREATE` request. The details of these attributes are described in Section 2.2.2.4. `JOB_CREATE` is the only request that is described using multiple lines. All the other requests can be described with a single line.

      A Ninf-G Client transfers a Request ID to Invoke Server. Invoke Server generates a unique Job ID and returns it to the Ninf-G Client. The Job ID is used by the Ninf-G Client to specify the job.

      When Invoke Server receives a `JOB_CREATE` request, it must send a Reply message to the Ninf-G Client. Then, Invoke Server generates a unique Job ID and notifies the Ninf-G Client of the Job ID. Finally, Invoke Server requests job invocation on remote servers via the underlying middleware used with the Invoke Server.

2. `JOB_STATUS`
    - Format

```
JOB_STATUS <Job ID><RET>
```

- Explanation

  This request queries Invoke Server on the status of jobs. The current version of Ninf-G4 and prior does not use this `JOB_STATUS` request.

3. `JOB_DESTROY`
   - Format

     ```
     JOB_DESTROY <Job ID><RET>
     ```

   - Explanation

     This request is used to terminate and destroy jobs. Invoke Server cancels all jobs if it receives this request and the corresponding jobs are not completed. When Invoke Server confirms that all jobs are cancelled, it sends `DONE` to the Ninf-G Client.

4. `EXIT`
   - Format

     ```
     EXIT<RET>
     ```

   - Explanation

     This request is used to terminate Invoke Server. If Invoke Server receives this `EXIT` request, it must cancel all outstanding jobs and wait for their termination.

2.2.2.2 Reply

Invoke Server must send a Reply message to a Ninf-G Client if Invoke Server receives a Request message from that Ninf-G Client.

The reply to `JOB_CREATE`, `JOB_DESTROY`, and `EXIT` messages is:

```
[S   | F <Error String>]<RET>
```

where `S` is sent in case of Success. Otherwise, `F` is returned, followed by <Error String>.

The reply to a `JOB_STATUS` message is:

```
[S <Status>  | F <Error String>]<RET>
```

Where <Status> is denoted as:

```
<Status> : [PENDING | ACTIVE | DONE | FAILED]
```

Each status indication indicates the status such that:

- `PENDING` : the Ninf-G Executable is waiting for invocation.
- `ACTIVE` : the Ninf-G Executable is already invoked.
- `DONE` : the Ninf-G Executable is already done.
- `FAILED` : the Ninf-G Executable exited abnormally.

2.2.2.3 Notify

A Notify message is used to send an asynchronous message from Invoke Server to a Ninf-G Client. Two types of Notify message are provided.

1. `CREATE_NOTIFY`
   - Format

     ```
     CREATE_NOTIFY <Request ID> [S <Job ID> | F <Error String>]<RET>
     ```

   - Explanation

     This is used to notify the Ninf-G Client of the Job ID. A Job ID is case sensitive and cannot include invisible characters.

2. `STATS_NOTIFY`
   - Format

     ```
     STATUS_NOTIFY <Job ID> <Status> <String><RET>
     ```

     ```
     <Status> : [PENDING | ACTIVE | DONE | FAILED]
     ```

   - Explanation

     This message is used to send notification that the status of a job has been changed.

     <String> can be any string, and the <String> is stored in an output log. It should be noted that the status of job can be changed from `PENDING` to `DONE`.

2.2.2.4 `JOB_CREATE` Request

This section describes the details of a `JOB_CREATE` Request.

- Format

  ```
  JOB_CREATE <Request ID><RET>
  hostname .....<RET>
  port .....<RET>
  ... (snip)
  JOB_CREATE_END<RET>
  ```

  Attributes are placed between `JOB_CREATE<RET>` and `JOB_CREATE_DONE<RET>`. Only one attribute can occupy one line and one line must include one and only one attribute. Attributes can be placed in any order. There are two types of attributes, mandatory attributes and optional attributes. Invoke Server must return an error if mandatory attributes are not included. Any unknown optional attributes must be ignored.

- Attributes

  The following is a list of attributes supported by Ninf-G. Some of these attributes are provided for the Globus Toolkit's Pre-WS GRAM and WS-GRAM. Any new attribute can be defined using the Client configuration file <SERVER> section "`invoke_server_option`" attribute.

name	mandatory	meanings
hostname	yes	Host name of the server
port	yes	Port number
jobmanager	no	Job Manager
subject	no	Subject of the GRAM
client name	yes	Host name of the Ninf-G Client
executable path	yes	Path of the Ninf-G Executable
backend	yes	Backend of the remote function (e.g., MPI)
count	yes	Number of Ninf-G Executables
staging	yes	A flag indicating if staging is used or not
argument	yes	Arguments for the Ninf-G Executable
work directory	no	Working directory of the remote function
gass url	no	The URL of GASS
redirect enable	yes	A flag indicating redirection of stdout/stderr
stdout file	no	file name of stdout
stderr file	no	file name of stderr
environment	no	Environment variables
tmp dir	no	temporary files directory
status polling	yes	Interval of status polling
refresh credential	yes	Interval of credential refresh
max time	no	Maximum execution time
max wall time	no	Maximum wall clock time
max cpu time	no	Maximum CPU time
queue name	no	Name of the queue
project	no	Name of the project
host count	no	Number of executables per host
min memory	no	Minimum size of requested memory
max memory	no	Maximum size of requested memory
rsl extensions	no	RSL extension

  Detailed description

  - hostname

    Host name of the server machine.

  - port

    The server port number on which the server is listening. The default value is depend on underlying middleware.

- jobmanager

  The job manager used on the server machine.

- subject [subject]

  The certificate subject of the resource manager contact.

- client_name [client name]

  The host name of the client machine.

- executable_path [path to the executable]

  Absolute path of the Ninf-G Executable. The path represents a remote path if staging is off. Otherwise, the path represents a local path.

- backend [backend]

  The method for launching the Ninf-G Executable is specified as backend. The value is `NORMAL`, `MPI`, or `BLACS`. If `MPI` or `BLACS` is specified, the Ninf-G Executable must be invoked via the mpirun command.

- count [N]

  The number of Ninf-G Executables to be invoked. If the backend is `MPI` or `BLACS`, count means the number of nodes.

- staging [true/false]

  The value is true if staging is on and Invoke Server must transfer the Ninf-G Executable file from the local machine to the remote machine.

- argument [argument]

  An argument for the Ninf-G Executable is specified using this attribute. This attribute can specify one argument only, and multiple arguments must be specified one by one, by using this attribute for each one. The arguments must be passed to the Ninf-G Executable as arguments.

  Example:

  ```
  argument --client=...
  argument --gass_server=...
  ```

- work_directory [directory]

  This attribute specifies the directory in which the Ninf-G Executable is invoked.

- gass_url

  This directory specifies the URL of the GASS server on the Client machine. This attribute is used for the Globus Toolkit's Pre-WS GRAM.

- redirect_enable [true/false]

  This attribute is set to true if the stdout/stderr of the Ninf-G Executable has been requested to be transferred to the Ninf-G Client.

- stdout_file [filename]

  If redirect_enable is set to true, this attribute specifies the name of the output file for stdout.

Invoke Server must output the stdout to this file. The Ninf-G Client reads this file as an output file and writes the contents of the file to the stdout of the Ninf-G Client.

- stderr_file [filename]

  If redirect_enable is set to true, this attribute specifies the name of the output file of the stderr. Invoke Server must output the stderr to this file. The Ninf-G Client reads this file as an output file and writes the contents of the file to the stderr of the Ninf-G Client.

- environment [ENV=VALUE]

  The environment variable for the Ninf-G Executable is passed using this attribute. The environment variable and its value are connected by =. Only the variable is specified if it does not take a value. Multiple environment variables must be specified one by one.

- tmp_dir [directory]

  The directory in which temporal files are placed.

- status_polling [interval]

  Invoke Server may need to check the status of jobs by polling the status of existing jobs. This attribute specifies the interval of the polling. The value is in seconds, and if it is not specified, the default value 0 is passed.

- refresh_credential [interval]

  This attribute specifies the interval for refreshing credentials. The value is in seconds, and if it is not specified, the default value 0 is passed.

- max_time [time]

  This attribute specifies the maximum time of the job.

- max_wall_time [time]

  This attributes specifies the maximum wall clock time of the job.

- max_cpu_time [time]

  This attribute specifies the maximum cpu time of the job.

- queue_name [queue]

  This attribute specifies the name of the queue to which the Ninf-G Executable should be submitted.

- project [projectname]

  This attribute specifies the name of the project.

- host_count [number of nodes]

  This attribute specifies the number of nodes.

- min_memory [memory size (MB)]

  This attribute specifies the minimum requirements for the memory size of the job.

- max_memory [memory size (MB)]

  This attribute specifies the maximum memory size of the job.

- rsl_extensions [RSL extension]

  This attribute can be used to specify the RSL extension which is available for the Globus Toolkit's WS GRAM.

Appendix A. How to specify the Invoke Server

Invoke Server is specified by the Ninf-G Client using a Client configuration file.

A.1. How to specify Invoke Server

Invoke Server is specified by using the invoke_server attribute in the <SERVER> section.

```
invoke_server [type]
```

Type specifies the type of the Invoke Server, such as GT4py or UNICORE.

A.2. How to pass information to Invoke Server

Invoke Server may require options for its execution. Such options can be specified by an option attribute in the <INVOKE_SERVER> section or by an invoke_server_option attribute in the <SERVER> section.

```
option [String]
invoke_server_option [String]
```

Multiple attributes can be specified in the <SERVER> or <INVOKE_SERVER> sections.

A.3. Polling interval

Invoke Server must check the status of jobs, and this may be implemented using polling. The polling interval can be specified by the status_polling attribute in the <INVOKE_SERVER> section.

```
status_polling [interval (seconds)]
```

A.4. Logfile

The filename of the Invoke Server's execution log can be specified by the invoke_server_log attribute in the <CLIENT> section.

```
invoke_server_log [filename]
```

If this attribute is specified, Invoke Server outputs logs to a file with the specified filename and file type of that Invoke Server.

The log_filePath attribute in the <INVOKE_SERVER> section can be used to specify a log file for a specific Invoke Server.

```
log_filePath [Log file name]
```

A.5. Maximum number of jobs per Invoke Server

The maximum number of jobs per Invoke Server can be limited by the max_jobs attribute in the

<INVOKE_SERVER> section. If the number of requested jobs exceeds this value, the Ninf-G Client invokes a new Invoke Server and requests that Invoke Server to manage the new jobs.

```
max_jobs [maximum number of jobs]
```

A.6. How to specify the path of the Invoke Server

If Invoke Server is not located in a pre-defined directory, the path attribute in <INVOKE_SERVER> can be used to specify the path of the Invoke Server.

```
path [path of the Invoke Server]
```

Appendix B. Miscellaneous Information

B.1. Job Timeout

The Job Timeout function is managed by the Ninf-G Client. Invoke Server is not responsible for the timeout.

B.2. Redirect stdout/stderr is implemented using files

Redirect stdout/stderr is implemented using files.

- The Ninf-G Client passes the filename to Invoke Server as an attribute for the `JOB_CREATE` request.
- Invoke Server outputs the stdout/stderr of the Ninf-G Executable to the file.
- The Ninf-G Client outputs the contents of the file to the stdout/stderr.

---

last update : $Date: 2006/09/20 04:57:42 $

11. Known problems

This section describes known problems in Ninf-G Version 4.2.5.

11.1 Problems related to the Globus Toolkit 4.2.

- 11.1.3 If GT4.2 is used, Invoke Server GT4py fails for invocation of remote processes.

  This problem is due to the bug of GT4.2.

  This problem can be avoided by configuring Java WS Core. Add "publishHostName" parameter to <globalConfiguration> in $GLOBUS_LOCATION/etc/globus_wsrf_core/server-config.wsdd. Set the value of "publishHostName" to "true".

- 11.1.2 Ninf-G Client and Executables may abort when they close connections.

  If Ninf-G Client is built with GT4.2, Ninf-G Client and Executables may abort when their connections are closed.

  This problem is due to the bug of GT4.2.

- 11.1.1 Ninf-G Client may abort when it queries to MDS4 server.

  If Ninf-G Client is built with GT4.2, Ninf-G Client may abort when it queries to MDS4 server.

This problem is due to the bug of GT4.2 (Bug# 6268)

11.2 Problems related to the Globus Toolkit

- ### 11.2.15 'authonly' for 'crypt' attribute is not available for Java Client.

  'authonly' is not available for Java Client.

  This problem is due to the bug of GT4 (Bug# 4669)

- ### 11.2.14 Ninf-G Client built as nonthread flavor sometimes freezes when 'crypt' attribute is set to 'authonly'.

  Ninf-G Client built as nonthread flavor sometimes freezes when 'crypt' attribute is set to 'authonly'.

  This problem is due to the bug of GT4 (Bug# 4354)

- ### 11.2.13 refresh_credential causes segmentation fault.

  Refresh credential causes Segmentation fault on GT4.

  This problem is due to the bug of the GT4 (Bug #4620).

- ### 11.2.12 Authentication only may cause freezes with Non Thread flavor.

  Authentication only may cause freezes with Non Thread flavor though authentication only works fine with Pthread flavor.

  This problem is due to the bug of GT4 (Bug# 4354).

- ### 11.2.11 Java Client does not work with authentication only.

  Java Client does not work with authentication only.

  This problem is due to the bug of GT4 (Bug# 4669).

- ### 11.2.10 WS GRAM Service on Mac OS X is not available for Ninf-G.

  Ninf-G Client cannot invoke Ninf-G executables via Invoke Server GT4py / WS GRAM on Mac OS X.

  This problem is due to the bug of the GT 4 (Bug# 4321).

- ### 11.2.9 Ninf-G Client returns 1 as an exit code when it is terminated by signal.

  Ninf-G Client returns an exit code 1 in the following case.

  - Ninf-G Client is terminated by a signal after grpc_finalize().
    and
  - The signal sent to the Ninf-G Client is either SIGINT, SIGTERM, or SIGHUP.

  This problem occurs only when Ninf-G Client uses pthread flavor of GT4. This is due to the bug of GT4 (Bug# 4287, 4294).

- ### 11.2.8 Invoke Server GT4py for GT 4.0.1 WS GRAM does not work correctly on Solaris.

  Initializing Function/Object handles using Invoke Server GT4py for GT 4.0.1 WS GRAM does not work correctly on Solaris.

  This problem is due to the bug of GT 4.0.1 (Bug# 4275).

- ### 11.2.7 Functions for initialization of function/object handles are MT-unsafe if Ninf-G is built with GT4.

  Functions for initialization of function/object handles (e.g. grpc_function_handle_init()) are MT-unsafe if Ninf-G is built with GT4 and Ninf-G Client does not use Invoke Server.

  Thus Ninf-G Client may not work correctly, if multiple threads initialize function/object handles simultaneously.

  This problem is due to the bug of GT4 (Bug# 3942).

  This problem has been fixed on Globus Toolkit Version 4.0.2 or later.

- ### 11.2.6 client_timeout does not work for MDS4 if it uses HTTP protocol.

  If client_timeout is specified for MDS4 with HTTP protocol, Ninf-G client will freeze when it will reach to the timeout time. This bug is due to the bug of GT 4.0.1 (Bug# 4157).

  By default, Ninf-G uses HTTPS protocol which does not cause this problem.

- ### 11.2.5 Ninf-G Client may cause segmentation faults on IA64 or AMD64.

  Ninf-G Client may cause segmentation faults if it meets the following conditions.

  - Version of the Globus Toolkit is 4.0.1.
  - SuSE Linux runs on IA64 or AMD64.
  - Ninf-G is configured with --with-mds2 option.

This problem is due to the bug of the GT 4.0.1 (Bug# 3921).

- 11.2.4 Java WS container can not accept many simultaneous RPCs.

  Java WS container causes "OutOfMemoryError" and outputs the following message when it receives many simultaneous requests for RPC.

  ```
          :
  ERROR container.ServiceThread
  [ServiceThread-11599,run:306] Run out of heap (server level)
  java.lang.OutOfMemoryError
  ERROR container.ServiceThread
  [ServiceThread-11551,doFault:701] Run out of memory (application level)
  java.lang.OutOfMemoryError
          :
  ```

  In order to avoid this problem, increase the maximum heap size of the JVM when running the container.
  It is described in the documentation of "GT 4.0: Java WS Core" , "4.1.2. Recommended JVM settings for the Java WS Core container".

- 11.2.3 Refresh credential does not work with GT4.0.1 WS GRAM.

  Due to the bug #3448, refresh credential does not work in GT4.0.1 WS GRAM. Bug is in ReliableFileTransferResource class and please refer ViewCVS for more details.

  - http://viewcvs.globus.org/viewcvs.cgi/ws-transfer/reliable/service/java/source/src/org/globus/transfer/reliable/service/ReliableFileTransferResource.java

- 11.2.2 Encryption may cause freezes with Non Thread flavor of GT3 or later.

  Encryption may cause freezes with Non Thread flavor of GT3 (Globus Toolkit Version 3) or later though encryption works fine with Pthread flavor of GT3 or later.

  This problem is due to the bug of GT3 or later (Bug# 4354).

- 11.2.1 Ninf-G Client freezes if an unknown host is specified as a server.

  Ninf-G Client freezes if it tries to create a function/object handle for an unknown host whose hostname can not be resolved. This problem is caused by the bug of the Globus Toolkit Version 2 library compiled with pthread flavor. The library compiled with non thread flavor does not cause this problem and the bug was fixed for the Globus Toolkit Version 3 or later.

  This problem has been fixed on Globus Toolkit Version 3.2 or later.

11.3 Problems related to environments (OS, Compiler, Architecture, etc.)

- 11.3.8 grpc_get_info_np() returns incorrect CPU time.

  grpc_get_info_np() returns incorrect CPU time if Ninf-G is built with pthread flavor on Solaris and Linux which use not NPTL but linuxthreads.

- 11.3.7 Ninf-G Client does not terminate if _exit() is called in a signal handler.

  Ninf-G Client does not terminate if _exit() is called in a signal handler. This problem occurs if Ninf-G Client is built with pthread flavor on Linux which uses not NPTL but linuxthreads.

  If the version of Linux kernel is 2.6 or later, thread library is usually NPTL and should not cause this problem.

  For glibc 2.3.2 or later, you can use getconf command to confirm the name of the thread library.

  ```
  % getconf GNU_LIBPTHREAD_VERSION
  linuxthreads-0.10
  ```

  Otherwise, the following command prints the name of the thread library.

  ```
  % `ldd /bin/ls | grep 'libc.so' | awk '{print $3}'` |¥
  egrep -i 'nptl|threads'
  ```

- 11.3.6 PGI compiler causes syntax-error on stub file generation from IDL.

  ng_gen command generates C source files for stub programs from IDL. Prior to generating the C source files, the IDL file is processed by C pre-processor (CPP). CPP command is detected when the Ninf-G is configured.

  PGI compiler may generate illegal C source files when it is used as CPP. Here is an example.

  ```
  Before CPP
  Globals { #include <stdio.h> }

  After CPP
  Globals { # include < stdio . h > }
  ```

  (Unexpected space characters are included before and after the period.)

  There are 2 ways to avoid this problem.

1. Use `--no-cpp` or `--with-cpp` option when executing ng_gen.

   ng_gen has options to skip CPP or to change CPP.

   ```
   % ng_gen --no-cpp target.idl
   ```

   `--no-cpp` option skips pre-processing and it is effective only when the pre-processor macros are not used in the IDL file.

   ```
   % ng_gen --with-cpp="gcc -xc -E" target.idl
   ```

   `--with-cpp` option specifies CPP command. In this example, macros in the IDL file are expanded by GCC pre-processor, but pre-processing and compilation of stub programs are performed by PGI compiler.

2. Modify the IDL file.

   For quotation of the file name, use double-quotes ("stdio.h") instead of less-than and greater-than (<stdio.h>).

- 11.3.5 Invoke Server GT4py on AIX 5.2 requires patch for Python.

  On AIX 5.2, Python does not work correctly due to a bug of Python. Ninf-G provides a patch for this bug and it is included in ng-4.x.x/external/python_aix_patch directory in the Ninf-G package. See README in the directory for more details.

- 11.3.4 Invoke Server GT4py doesn't work with /usr/bin/python on Mac OS X.

  Invoke Server GT4py will exit immediately if /usr/bin/python is used on Mac OS X. This problem can be avoided by using python 2.4.2 built from source bundle.

- 11.3.3 Assembler may detect warnings when optimization is enabled.

  Assembler may detect warnings when optimization is enabled with GCC Version 3.2.2 or prior on Itanium2.
  The warning message is as follows:

  ```
  cc -Wall -O2 <snip> -c -o ngclSession.o ngclSession.c
  /tmp/ccMDS56n.s: Assembler messages:
  /tmp/ccMDS56n.s:10125: Warning: Use of 'mov' may violate WAW dependency 'GR%, %in 1 - 127' (impliedf), specific resource number is 14
  /tmp/ccMDS56n.s:10124: Warning: This is the location of the conflicting usage
  /tmp/ccMDS56n.s:10396: Warning: Use of 'mov' may violate WAW dependency 'GR%, %in 1 - 127' (impliedf), specific resource number is 14
  /tmp/ccMDS56n.s:10395: Warning: This is the location of the conflicting usage

  cc -Wall -O2 <snip> -c -o ngStream.o ngStream.c
  /tmp/cc35sx1T.s: Assembler messages:
  /tmp/cc35sx1T.s:2893: Warning: Use of 'mov' may violate WAW dependency 'GR%, %in 1 - 127' (impliedf), specific resource number is 14
  /tmp/cc35sx1T.s:2892: Warning: This is the location of the conflicting usage
  ```

  This problem is reported to GCC bugzilla.
  (http://gcc.gnu.org/bugzilla/show_bug.cgi?id=7908)
  This problem has been fixed on GCC Version later than 3.2.2.

- 11.3.2 Java Client sometimes fails when running large-scale applications on Windows.

  On Windows, ServerSocket sometimes fails to accept requests for connection from Ninf-G Executables if it receives many (about 10 or more) requests at the same time.
  This should be an Windows-specific problem since it does not appear on Linux or Solaris.
  Use Linux or Solaris for large-scale applications.

- 11.3.1 dcomplex type is not available with gcc-2.96 on IA64 platform.

  dcomplex variables is not available if gcc-2.96 is used on IA64 platform.
  Although compilation of Ninf-G Client program will be succeed, the Ninf-G client program will be terminated with an error.
  In order to avoid this problem, other version of gcc should be used.

11.3 Problems related to flavors (especially for non-thread flavor)

- 11.4.4 Invoke Server is not supported for non-thread flavor.

  The Invoke Server feature in Ninf-G Client is not supported for non-thread flavor.

- 11.4.3 Job start timeout feature is not completely supported for non-thread flavor.

  Job start timeout feature (in <SERVER> section in Client configuration file and handle attributes) does not work with non-thread flavor in the following case.

  - The first RPC after initializing the function handle is called with setting argument_transfer attribute or session attribute to copy or nowait.
- 11.4.2 Refresh credentials and Session timeout feature is not supported for non-thread flavor.

  The Refresh credentials (in <CLIENT> section in client configuration file) and Session timeout (RPC timeout) (in <FUNCTION_INFO> section in client configuration file) feature in Ninf-G Client is not supported for non-thread flavor.

- 11.4.1 heartbeat detection by Ninf-G Client is not supported for non-thread flavor.

  Heartbeat detection by Ninf-G Client is not supported for non-thread flavor. In order to detect the heartbeat from Ninf-G Executables, use pthread flavor for Ninf-G Client.

Note: Sending heartbeat by Ninf-G Executables is supported for both pthread and non-thread flavors.

11.4 Problems related to version compatibility

- 11.5.2 "TCP connect retry" function is not supported by Ninf-G 2.3.0 and prior versions.

  Ninf-G 2.4.0 provides a new capability to retry to establish a TCP connection between the Ninf-G Executable and the Ninf-G Client if the Ninf-G Executable fails to connect to the Ninf-G Client.

  This is a new capability which is not supported by Ninf-G 2.3.0 and prior versions. Although this capability will be provided in the future release of Ninf-G, be sure that it will cause a problem if the version of Ninf-G on the client is 2.4.0 or later and the version of Ninf-G on the server is 2.3.0 or prior.

  This problem can be avoided by setting tcp_connect_retryCount to 0 on the <SERVER> section of client configuration file.

- 11.5.1 backend=MPI may cause error between different versions of Ninf-G.

  Using different versions of Ninf-G on a client and servers may cause an error if MPI is used as backend of a remote executable.
  It is recommended to use the same version of Ninf-G on a client and servers.

11.5 Problems related to NAREGI SS

- 11.6.1 Ninf-G Executable invoked using Invoke Server NAREGISS may not be able to create temporary files.

  When the job is invoked by the function for initializing an array of function/object handles(grpc_function_array_handle_np() etc.) and Invoke Server NAREGISS is used, environment variable TMPDIR in the Ninf-G Executable may be set to directory which does not exist. Thus, Ninf-G Executable may not be able to create temporary files.

  This problem can be avoided by specifying "tmp_dir" attribute in Ninf-G Executable configuration file.

11.6 Other problems

- 11.7.2 The hostname attributes in <CLIENT> section in the client configuration file may not be set appropriately.

  The hostname attribute in <CLIENT> section in the client configuration file may not be set appropriately if globus_libc_gethostname() is called before grpc_initialize().

- 11.7.1 Ninf-G Executable may not be died if MPI is specified as backend.

  When a Ninf-G Client is terminated by SIGTERM or SIGINT, Ninf-G Executable may not be died if MPI is specified as backend.
  If you want to use MPI and terminate a Ninf-G Client by SIGTERM or SIGINT, please use Non Thread flavor at remote side. However, Ninf-G Executable may not be died until a remote function is completed.

---

last update : $Date: 2008/10/14 03:52:29 $