Описание компонента JobManager библиотеки классов GridMD

1 Определения

1.1 Используемые термины

- **Система управления заданиями** – внешний программный пакет, установленный на головной машине кластера или Грида, обеспечивающий выделение ресурсов, запуск и дальнейшее управление программами пользователей на данной вычислительной системе. Примерами систем управления заданиями являются PBS, SLURN, Globus, Condor и др.
- **Удаленная система (удаленный ресурс)** – компьютер, на котором установлена система управления заданиями.
- **Задание** – минимальный объект, под который система управления заданиями выделяет ресурсы. Обычно задание включает в себя исполняемое приложение, набор входных данных и параметры запуска.
- **Менеджер заданий** – объект класса, наследованного от gmJobManager (см. ниже), на основе которого реализован пользовательский интерфейс GridMD к функциям той или иной системы управления заданиями.

1.2 Использование внешних библиотек

Компонент JobManager использует объектно-ориентированную библиотеку классов wxWidgets (http://www.wxwidgets.org), распространяемую по лицензии L-GPL. Используется только модуль Base. Требуемая версия библиотеки 2.8.12 и выше. В интерфейсе пользователя используются классы строк wxString и массива строк wxArrayString

1.3 Дополнительных типы данных, используемые при описании функций JobManager

```
typedef char* pSTR;
typedef const char* pCSTR;
typedef list<gmJob*> JobList;
typedef std::map<wxString,wxString> gmExtraParHash;
```

1.4 Дерево классов

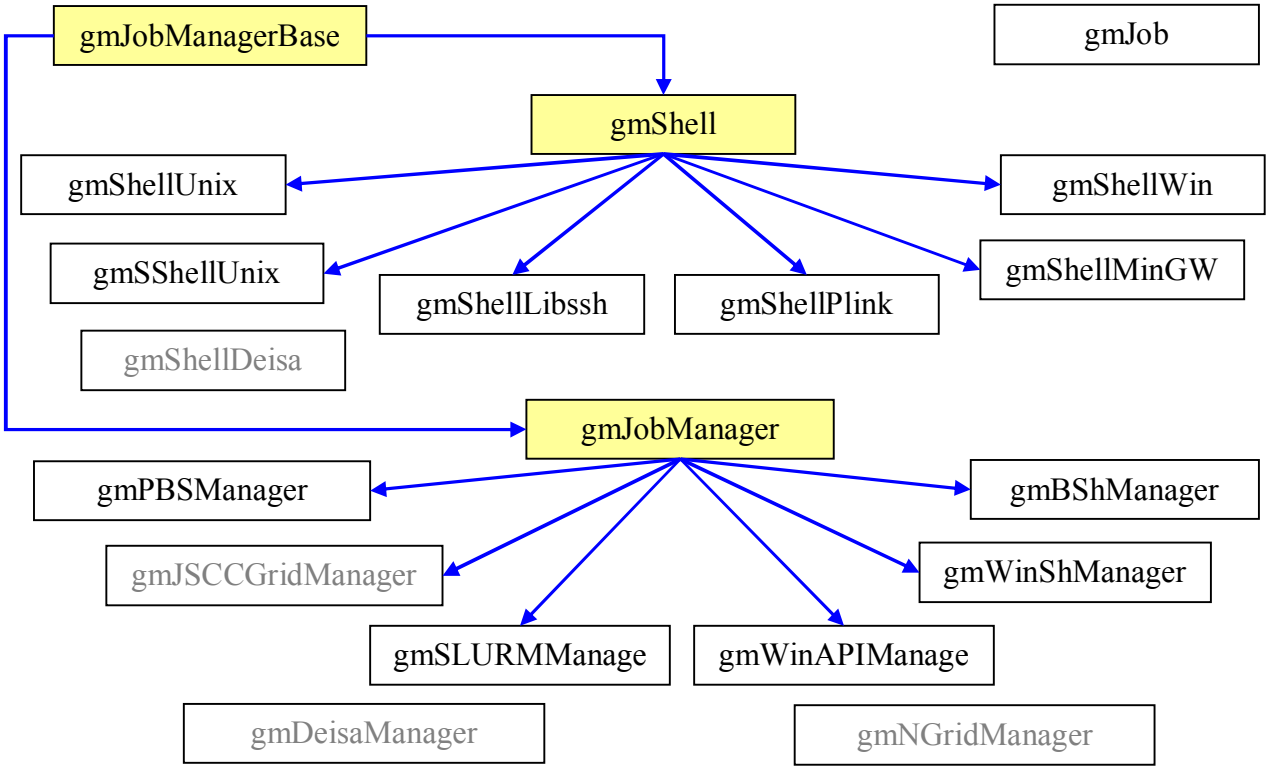

Рис. 1. Структура классов в объектно-ориентированной модели менеджера заданий GridMD. Желтой заливкой показаны абстрактные классы. Серым шрифтом показаны классы не поддерживаемые текущей версией GridMD (устаревшие или имеющие ограниченную функциональность).

2 Описание функций библиотеки

2.1 Выбор протокола доступа к удаленной системе

Для работы с любой внешней системой управления заданиями необходимо в первую очередь выбрать протокол доступа к машине, на которой эта система установлена. Данную машину мы будем называть «удаленным ресурсом», хотя в некоторых случаях она может представлять собой тот же компьютер, на котором запускается GridMD-приложение. Каждому протоколу доступа соответствует один из описанных ниже классов, наследуемых от базового класса gmShell (см. рис. 1).

```
gmShellPlink(pCSTR login = "", pCSTR host = "", pCSTR plink_args = "")
// jobmngr/plinkshell.h
```

Служит для выполнения приложения GridMD под ОС Windows и обращения к удаленному серверу, работающему под управлением ОС Linux. Выполнение удаленных команд (bash shell) и копирование файлов производится по протоколу SSH с применением утилит plink.exe и pscp.exe из пакета PuTTY (http://www.chiark.greenend.org.uk/~sgtatham/putty/). Параметры конструктора: *login* – имя пользователя, *host* – имя или IP адрес удаленного узла, *plink_args* – дополнительные аргументы командной строки для запуска plink.exe и pscp.exe (см. описание на сайте проекта PuTTY). Перед первым использованием протокола gmShellPlink в *plink_args* необходимо задать один из вариантов авторизации, не требующей интерактивного ввода пароля:

```
-pw <password>          - пароль указывается в явном виде (небезопасно)
```

```
-load "<session_name>"  - настройки соединения загружаются из сохраненной
                          сесиии PuTTY
-i <ssh_key_path>       - используется файл с закрытым ключом
```

Аргументы "-ssh -batch" добавляются автоматически, и их не следует указывать в *plink_args*.

```
gmShellUnix()  // jobmngr/unixshell.h
```

При выборе этого протокола доступа предполагается, что приложение GridMD запускается под ОС Unix/Linux на том же узле, где установлена система управления заданиями. Для выполнения команд используется локальный вызов bash shell.

```
gmSShellUnix(pCSTR login = "", pCSTR host = "", pCSTR ssh_args = "")
// jobmngr/unixsshell.h
```

При выборе этого протокола доступа предполагается, что приложение GridMD запускается под ОС Unix/Linux, доступ к удаленной системе и копирование файлов осуществляется с помощью команд ssh, scp. Значения параметров аналогичны gmShellPlink.

```
gmShellLibssh(pCSTR login = "", pCSTR host = "")
// jobmngr/libsshell.h
```

Приложение GridMD может запускаться под ОС Unix/Linux или Windows, доступ к удаленной системе и копирование файлов осуществляется по протоколу SSH/SCP с использованием функций библиотеки с открытым кодом LibSSH (http://www.libssh.org). Данный протокол обеспечивает меньшую задержку по сравнению с gmShellPlink, gmSShellUnix при выполнении команды на удаленном узле или копировании файлов, так как не требует открытия новой сессии SSH для каждой из указанных операций.

```
gmShellWin()  // jobmngr/winshell.h
```

Приложение GridMD выполняется под ОС Windows, команды пользователя выполняются на том же компьютере путем фонового запуска batch-файлов Windows, управление заданиями и операции с файлами – с использованием функций Win32 API.

```
gmShellMinGW(pCSTR bash_path = "bash.exe", pCSTR home_path="")
// jobmngr/mingwshell.h
```

Приложение GridMD выполняется под ОС Windows, команды пользователя выполняются на том же компьютере с использованием пакета Minimalist GNU for Windows (MinGW). Пакет MinGW и его компонент MSYS (http://www.mingw.org/wiki/MSYS), должны быть предварительно установлены пользователем. Команды выполняются либо путем создания batch-файлов, либо путем выполнения скриптов в bash shell из пакета MinGW MSYS. В параметре *bash_path* указывается путь к исполняемому файлу bash.exe из пакета MinGW, а в *home_path* – домашняя директория пользователя в MinGW MSYS.

2.2 Настройка параметров протокола доступа

Параметры проколов доступа, описанные в предыдущем разделе, настраиваются с помощью функций базового класса:

```
void SetParam(pCSTR par_name, pCSTR par_value)
void SetParam(pCSTR par_name, int par_value)
void SetParam(pCSTR par_name, double par_value)
```

Данные функции принимают название параметра в текстовом виде *par_name* и его значение *par_value* типа pCSTR, int или double. Фактически все значения параметров типа int и double также преобразуются в строку и хранятся в текстовом виде. Получить текущее значение любого параметра можно функцией:

```
wxString GetParam(pCSTR par_name)
```

Пользователь может использовать только имена параметров, определенные для данного класса. Список всех доступных параметров, их типов, описаний и значений по умолчанию, приведен в таблице 1. Тип параметра, указанный во втором столбце таблицы, означает, что текстовое представление данного параметра должно конвертироваться в значение соответствующего типа. Например, параметр plink_att_num типа int может быть задан любой из функций:

```
shell.SetParam("plink_att_num", "3");
shell.SetParam("plink_att_num", 3);
shell.SetParam("plink_att_num", 3.0);
```

где shell – объект соответствующего класса. Для параметров типа bool допустимыми значениями являются “0”, “false” или “1”, “true”.

Таблица 1. Описание настраиваемых параметров для классов протокола доступа.

Название параметра	**Тип, дипазон**	**Значение по умолчанию**	**Описание**
Все классы протоколов доступа			
rem_tmp_dir*	string	.gmJobManager	Каталог на удаленной системе, в котором будут создаваться рабочие подкаталоги для заданий
home_dir	string		Путь к домашнему каталогу пользователя на удаленной системе. Если этот параметр не задан, т.е. переменная содержит пустую строку, при первом обращении к удаленной машине в нее будет записано значение домашнего каталога пользователя.
home_dir_win	string		Путь к домашнему каталогу пользователя на локальном компьютере, работающем под управлением ОС Windows (в обозначениях Windows)
rem_perm_dir	string		Если это поле не пусто, то в адресах файлов на удаленной системе последовательность $PERM_STORAGE будет заменена на содержимое данного параметра. Предполагается, что таким образом можно задавать каталог для постоянного хранения файлов на удаленной системе
local_remove	bool	false	Если =false, то удаление файлов на локальной системе функцией RemoveLocal запрещено
dump_commands	bool	false	Вывод всех исполняемых на удаленной системе команд в stdout для отладки
Класс gmShellUnix *(выполнение команд на локальной Unix-машине)*			
Нет дополнительных параметров			
Класс gmSShellUnix *(выполнение команд на Unix-машине с доступом по SSH)*			
host	string		Имя или IP адрес удаленного узла
login	string		Имя пользователя на удаленном узле
ssh_args	string		Аргументы командной строки для ssh и scp
ssh_path	string	ssh	Команда запуска ssh на локальной машине
scp_path	string	scp	Команда запуска scp на локальной машине
Класс gmShellPlink *(доступ к Linux системе с помощью PuTTY)*			
host	string		Имя или IP адрес удаленного узла
login	string		Имя пользователя на удаленном узле
plink_args	string		Аргументы командной строки для программ plink.exe и pscp.exe
plink_path**	string	*см. ниже*	Путь к файлу plink.exe на локальной машине
pscp_path**	string	*см. ниже*	Путь к файлу pscp.exe на локальной машине
plink_att_num	int, >= 1	1	Количество попыток соединиться с удаленной системой при ошибке сетевого доступа

plink_retry_delay	int, >= 0	3000	Интервал между последовательными попытками соединения с удаленной системой (в мс)
Класс gmShellLibssh *(доступ по протоколу SSH с помощью библиотеки libssh)*			
host	string		Имя или IP адрес удаленного узла
login	string		Имя пользователя на удаленном узле
port	int, 1–65535	22	Порт для доступа по протоколу SSH к удаленному узлу
knownhosts	string		Путь к файлу “knownhosts”. Если не задан, то используется “$HOME/.ssh/knownhosts”.
host_accept	string	known	Соединение устанавливается, только если открытый ключ удаленного узла (host key) удовлетворяет требованиям, заданным значением данного параметра: *known* – только если ключ узла указан в файле “knownhosts” (т.е. подтвержден), *new* – если ключ подтвержден или является новым, *changed* – если ключ подтвержден, является новым или изменен по сравнению с “knownhosts”, *all* – во всех случаях.
password	string		Пароль для авторизации пользователя. Если не задан, выполняется авторизация с помощью открытого и закрытого ключей (см. ниже).
privkey	string		Путь к файлу с секретным ключом пользователя (private key) в формате OpenSSL. Если не задан, используются файлы “id_dsa” или “id_rsa” из каталога “$HOME/.ssh”
privkey_pass	string		Пароль для дешифровки секретного ключа пользователя (если ключ зашифрован)
log_verbosity	int, 0 – 4	0	Уровень подробности диагностических сообщений LibSSH: 0 – сообщения отключены, 4 - максимальная детализация
ssh_timeout	int, >=0		Максимальная задержка при установлении соединения с удаленным узлом в секундах (см. опцию “SSH_OPTIONS_TIMEOUT” в описании LibSSH). Если не задан, используется стандартное значение для данной версии библиотеки LibSSH.
conn_att_num	int, >= 1	3	Количество попыток соединиться с удаленной системой при ошибке сетевого доступа
conn_retry_delay	int, >= 0	3000	Интервал между последовательными попытками соединения с удаленной системой (в мс)
Класс gmShellWin *(выполнение batch файлов на локальной Windows-машине)*			
rem_tmp_dir	string	$(TEMPDIR)/ gmJobManager	Параметр имеет тот же смысл, что и для всех протоколов, изменено значение по умолчанию
rem_perm_dir	string		Параметр имеет тот же смысл, что и для всех протоколов, изменено значение по умолчанию
Класс gmShellMinGW *(выполнение команд на локальной Windows-машине с помощью MinGW)*			
rem_tmp_dir	string	gmJobManager	Каталог в системе MinGW, в котором будут создаваться рабочие подкаталоги для заданий
home_dir	string		Путь к домашнему каталогу пользователя в системе MinGW в обозначениях MinGW (например, “/home/username”)
home_dir_win	string		Путь к домашнему каталогу пользователя в системе MinGW в обозначениях Windows (например, “c:\\msys\\1.0\\home\\username”)
bash_path	string	bash.exe	Путь к исполняемому файлу bash.exe из пакета MinGW MSYS

* Если путь к файлу или каталогу не начинается с символа “/”, этот путь рассматривается как относительный к точке входа в удаленную систему (обычно точка входа – это домашний каталог).
** Если параметры plink_path, pscp_path на заданы, исполняемые файлы plink.exe и pscp.exe ищутся в каталогах "C:\Program Files\PuTTY " и "C:\Program Files (x86)\PuTTY ".

2.3 Операции с файлами и каталогами

Во всех классах протоколов доступа реализованы следующие сервисные функции, доступные пользователю:

```
int gmShell::MkDir(pCSTR remdir)
```

Создание каталога на удаленном ресурсе. Для Unix/Linux удаленных систем создаются все необходимые родительские каталоги (используется “mkdir -p”). Функция возвращает 0 в случае успешного выполнения.

```
int gmShell::MkDirLocal(pCSTR locpath)
```

Создание каталога на локальном компьютере. Функция возвращает 0 в случае успешного выполнения.

```
int gmShell::StageIn( pCSTR locpath, pCSTR rempath, unsigned flags = 0)
int gmShell::StageOut(pCSTR locpath, pCSTR rempath, unsigned flags = 0)
```

Копирование локального файла/каталога *locpath* на удаленный ресурс в файл/каталог *rempath* (StageIn) или копирование файла/каталога *remfile* с удаленного ресурса на локальный компьютер в файл/каталог *locpath* (StageOut). Если путь *rempath* не является абсолютным, т.е. не начинается с “/”, он рассматривается относительно каталога, в который пользователь попадает по умолчанию при входе на удаленную систему. Обычно, это домашний каталога пользователя. В пути к файлам на удаленной системе допускается явное указание домашнего каталога с помощью символа “~”, а также каталога для постоянного хранения файлов, определенного значением параметра “rem_perm_dir”, с помощью последовательности “$PERM_STORAGE”. Аргумент *flags* является битовым полем, задающим параметры копирования:

gmShell::MOVE	удаление исходного файла после копирования;
gmShell::TEXT	преобразование символов конца строки (LF/CR+LF), в соответствии с типом локальной и удаленной файловых систем (Windows/Unix);
gmShell::RECURSIVE	рекурсивное копирование каталогов;
gmShell::CREATEPATH	автоматическое создание выходного каталога;
gmShell::MAYNOTEXIST	игнорирование отсутствия входных файлов.

Использование флага gmShell::TEXT совместно с gmShell::RECURSIVE поддерживается не всеми классами протоколов доступа.

Пример использования:

```
rc = StageIn(“c:\file.txt”, “~/newdir/file1.txt”,
             gmShell::MOVE | gmShell::TEXT | gmShell::CREATEPATH )
```

В данном примере, исходный файл “c:\file.txt” будет скопирован в “~/newdir/file1.txt” на удаленной системе с автоматическим созданием каталога “~/newdir”. При использовании опции gmShell::CREATEPATH добавление символа “/” или “\” в конец пути-получателя:

```
rc = StageIn(“c:\file.txt”, “newdir/test/”)
```

приведет к тому, что файл “c:\file.txt” будет скопирован в “newdir/test/file.txt” с автоматическим созданием каталога “newdir/test”. Если исходный путь содержит маску или используется опция gmShell::RECURSIVE:

```
rc = StageIn("dir\*.txt", "newdir/test", gmShell::CREATEPATH)
rc = StageIn("dir", "newdir/test", gmShell::RECURSIVE | gmShell::CREATEPATH)
```

путь "newdir/test" всегда рассматривается как имя каталога-получателя, т.е. на удаленной системе будут получены файлы "newdir/test/*.txt" и каталог "newdir/test/dir". Аналогичные правила действуют для функции StageOut().

Функции возвращают 0 в случае успешного выполнения.

```
int gmShell::StageIn( const wxArrayString& locfiles, pCSTR remdir,
                      unsigned flags = 0)
int gmShell::StageOut(pCSTR locdir, const wxArrayString& remfiles,
                      unsigned flags = 0);
```

Копирование локальных файлов/каталогов, перечисленных в массиве *locfiles*, в каталог *remfile* на удаленном ресурсе (StageIn) или копирование удаленных файлов/каталогов *remfiles* в локальный каталог *locdir* (StageOut). Описание класса wxArrayString см. в документации к пакеку wxWidgets: http://docs.wxwidgets.org/stable/wx_wxarraystring.html#wxarraystring. Аналогично предыдущей паре функций, если путь к удаленным файлам и каталогам не является абсолютным, он рассматривается относительно каталога по умолчанию. Имена удаленных файлов и каталогов могут содержать символы "~" и "$PERM_STORAGE". В именах исходных файлов и каталогов могут использоваться маски "*" и "?", однако использование масок совместно с флагом gmShell::RECURSIVE поддерживается не всеми классами протокола доступа, а результат такого копирования может зависеть от типа локальной ОС и выбранного протокола. Функции возвращают 0 в случае успешного выполнения.

```
int gmShell::Remove(pCSTR remfile)
int gmShell::Remove(const wxArrayString& remfiles)
```

Удаление файла или каталога (рекурсивно) на удаленном ресурсе. Во втором варианте функции поочередно удаляются все файлы/каталоги из массива *remfiles*. Функции возвращают 0 в случае успешного выполнения.

```
int gmShell::RemoveLocal(pCSTR locpath)
```

Удаление файла или каталога (рекурсивно) на локальном компьютере. Если параметр "local_remove" имеет значение false, то функция всегда генерирует ошибку. Функции возвращают 0 в случае успешного выполнения.

```
int gmShell::Execute(const wxString& cmd)
int gmShell::Execute(const wxString& cmd, wxString& out, wxString& err)
```

Выполнение команды *cmd* на удаленном сервере. Во втором варианте вывод в потоки stdout и stderr будут перехвачены и сохранены в виде строк *out* и *err*. Возвращаемое значение – код завершения запущенной команды.

```
int gmShell::ExecuteAsync(const wxString& cmd)
```

Запуск команды *cmd* в фоновом режиме с потерей вывода в *out* и *err*. Реализована лишь для некоторых протоколов доступа (например, gmShellWin, gmShellMinGW). В случае успешного выполнения возвращает положительное число – идентификатор процесса (PID).

```
long gmShell::ExecTime()
long gmShell::TransferTime()
```

Первая функция возвращает суммарное время в миллисекундах, потраченное на выполнение команд на удаленной системе, вторая – суммарное время копирования файлов.

2.4 Инициализация и настройка менеджера заданий

Для инициализации менеджера заданий создается экземпляр одного из указанных ниже классов, наследованных от gmJobManager. Каждый класс соответствует то или иной системе управления заданиями. Первым аргументом для конструкторов всех классов является ссылка на созданный ранее класс протокола доступа, наследованный от gmShell.

```
gmPBSManager(gmShell& shell)  // определен в файле jobmngr/pbsmngr.h
```
Запуск задач на удаленной системе с помощью системы очередей Portable Batch System (PBS).

```
gmSLURMManager(gmShell& shell)  // определен в файле jobmngr/slurmmngr.h
```
Запуск задач на удаленной системе с помощью системы очередей Simple Linux Utility for Resource Management (SLURM).

```
gmBShManager(gmShell& shell)  // определен в файле jobmngr/bshmngr.h
```
Выполнение скрипта в bash shell на удаленной системе. Скрипт запускается в фоновом режиме с использованием утилиты nohup, которая позволяет не завершать процесс при разрыве сетевого соединения.

```
gmWinShManager(gmShell& shell)  // определен в файле jobmngr/winshmngr.h
```
Выполнение скрипта в bash shell или batch-файла Windows на локальной машине в среде MinGW. Предназначен для использования с классом протокола доступа gmShellMinGW.

```
gmWinAPIManager(gmShell& shell)  // определен в файле jobmngr/ winapimngr.h
```
Выполнение batch-файла Windows на локальной машине (в фоновом режиме). Предназначен для использования с классом протокола доступа gmShellWin.

Для всех классов менеджеров заданий можно использовать конструктор по умолчанию, а затем вызвать функцию Open, принимающую аргумент *shell*:

```
gmPBSManager mngr();
mngr.Open(shell);
```

Для всех классов менеджеров заданий определены следующие информационные функции:

```
gmShell* gmJobManager::GetShell() const
```
Возвращает указатель на выбранный протокол доступа

```
bool gmJobManager::mpi_support () const
```
Указывает, поддерживает ли менеджер выполнение MPI-заданий.

```
int gmJobManager::required_cmd_set() const
```
Возвращает тип набора команд, который долж

Каждый менеджер заданий требует, чтобы удаленная система поддерживает тот или иной набор команд (bash shell или batch-файлы Windows), поэтому он может работать только с совместимыми с ним протоколами доступа. Таблица совместимости классов менеджеров заданий и классов протокола доступа приведена в таблице 2.

Настройка дополнительных параметров менеджеров заданий производится так же, как и для протоколов доступа, т.е. с помощью функции SetParam (см. раздел 2.2). Описание параметров приведено в таб. 3.

Таблица 2. Совместимость классов менеджеров заданий и классов протокола доступа.

Shell / Job Manager	gmShell Unix	gmSShell Unix	gmShell Plink	gmShell Libssh	gmShell MinGW	gmShell Win	gmShell Deisa
gmPBSManager	+	+	+	+	+	–	–
gmSLURMManager	+	+	+	+	+	–	–
gmJSCCGridManager	+	+	+	+	+	–	–
gmNGridManager	+	+	+	+	+	–	–
gmBShManager	+	+	+	+	+	–	–
gmWinShManager	–	–	–	–	+	–	–
gmWinAPIManager	–	–	–	–	–	+	–
gmDeisaManager	–	–	–	–	–	–	+

Таблица 3. Описание настраиваемых параметров для классов менеджеров заданий.

Название параметра	**Тип, дипазон**	**Значение по умолчанию**	**Описание**
Все классы менеджеров заданий			
init_job_cmd	string		Набор команд, который будет передан в систему очередей и выполнен перед исполнением основных команд любого задания. Для MPI-заданий команды из init_job_cmd выполняются в последовательном режиме (до вызова mpiexec).
end_job_cmd	string		Набор команд, который будет передан в систему очередей и выполнен после исполнения основных команд любого задания. Для MPI-заданий команды из end_job_cmd выполняются в последовательном режиме (после вызова mpiexec).
pre_subm_cmd	string		Набор команд, который будет выполнен перед постановкой в задания очередь на головном узле удаленной системы. Вывод в stdout будет потерян, любой вывод в stderr или ненулевой код возврата приводит к отмене постановки задачи в очередь и переводу ее в состояние JOB_FAILED.
jobs_limit	int	0	Максимальное число одновременно запущенных заданий для данного менеджера. При превышении этого параметра задачи не ставятся в очередь, а переводятся в состояние JOB_PREPARED. Если =0, то количество запущенных заданий не ограничено, если <0, то все задания при выполнении функции Submit будут переводится в состояние JOB_PREPARED.
wait_timeout	int, >=0	0	Максимальное время ожидания в функциях Wait и WaitAll (в мс). Если =0, то время ожидания не ограничено.
wait_delay	int, >=0	3000/ 500(gmWinAPIManager)	Интервал, с которым функции ожидания Wait и WaitAll проверяют состояние задания на удаленной системе (в мс).
mpi_enabled	bool	true	Возможность запуска MPI-заданий. Если =false, то

			функция Sumbit для задания, с установленным атрибутом gmJob::mpi, генерирует ошибку
save_job_info	bool	false	Если =true, то в директорию со служебными файлами для данного задания (<rem_temp_dir>/job-<имя задачи>) записывается файл info с подробной информацией о задании
Класс gmPBSManager *(выполнение в системе очередей PBS)*			
mpi_run_cmd	string	mpiexec	Команда для запуска MPI-программы (используется, если установлен флаг gmJob::mpi).
qsub_args	string		Дополнительные аргументы команды qsub
pbs_path	string		Каталог с исполняемыми файлами qsub, qstat, qdel
job_name_prefix	string		Дополнительный префикс для имени задания, передаваемого в ключе –N команды qsub
Класс gmSLURMManager *(выполнение в системе очередей SLURM)*			
slurm_cmd_serial	string	srun	Команда с аргументами для постановки задания в очередь. Используется внутри скрипта, передаваемого утилите sbatch.
slurm_cmd_mpi	string	srun	То же, что и slurm_cmd_serial, но для MPI-задач
sbatch_args	string		Дополнительные аргументы команды sbatch
slurm_path	string		Каталог с исполняемым файлом sbatch
job_name_prefix	string		Дополнительный префикс для имени задания, передаваемого в ключе –J команды sbatch.
Класс gmBShManager *(выполнение команд в bash shell с использованием утилиты nohup)*			
Нет дополнительных параметров			
Класс gmWinShManager *(Выполнение batch-файла или скрипта в bash shell с помощью MinGW)*			
script_type	string	windows	Тип исполняемых скриптов: “windows” – batch файлы Windows, “bash” – скрипты bash shell
stat_read_ntries	int, >=1	20	Количество попыток прочитать PID из служебного файла stat после запуска задания
stat_read_delay	int, >=0	200	Интервал между попытками прочитать PID из служебного файла stat после запуска задания (мс)
Класс gmWinAPIManager *(Выполнение batch-файла средствами Win32 API)*			
kill_ntries	int, >=1	20	Количество попыток определить, что процесс завершен, после выполнения функции Stop
kill_delay	int, >=0	100	Интервал между попытками определить, что процесс завершен (в мс)

2.5 Создание задания

Для каждого задания необходимо создать отдельный объект класса gmJob. Такой объект может быть создан двумя способами. В первом случае пользователь создает его в динамической памяти или в стеке, используя конструктор:в

```
gmJob()
```

Такой объект считается «неуправляемым» (unmanaged) и за освобождение памяти, занятой объектом, отвечает пользователь.

Альтернативой этому является создание «управляемого» (managed) объекта с помощью функции менеджера заданий:

```
gmJob* gmJobManager::CreateJob()
```

Этот объект создается в динамической памяти и будет автоматически удален при выполнении функций Clear, Detach (см ниже) или при удалении менеджера заданий. Управляемые объекты изначально привязывается к породившему их менеджеру заданий и не могут быть запущены с помощью другого менеджера.

2.6 Подготовка задания к выполнению

Параметры задания задаются с помощью следующих членов класса gmJob.

```
void gmJob::AddInFile(pCSTR src, pCSTR workpath = "", unsigned flags = 0)
```

Функция определяет имя входного файла или каталога *src* который при выполнении функции Submit (см. ниже) будет скопирован во временную директорию для данного задания. Файл *src* может располагаться на локальной системе или на той же удаленной системе, на которой будет запускаться задание (см. ниже опцию gmJob::REMOTE). Аргумент *workpath* может указывать имя выходного файла и/или подкаталог относительно временной директории, если *workpath* – пустая строка, то файл/каталог копируются в корень временной директории с тем же именем. Аргумент *flags* является битовым полем, задающим параметры копирования:

gmJob::MOVE	удаление исходного файла/каталога *src* после копирования;
gmJob::TEXT	преобразование символов конца строки (LF/CR+LF), в соответствии с типом локальной и удаленной файловых систем (Windows/Unix);
gmJob::RECURSIVE	рекурсивное копирование каталогов;
gmJob::REMOTE	указывает, что исходный файл/каталог *src* находится на удаленной системе, на которой будет запущено задание;
gmJob::CREATEPATH	если во временной директории не существует выходного каталога с именем, указанным в *workpath*, он будет создан автоматически;
gmJob::MAYNOTEXIST	игнорирование отсутствия входных файлов.

Параметры объединяются в аргументе *flags* с помощью операции "|", например:

```
job.AddInFile("localdir/file", "outdir/", gmJob::TEXT | gmJob::CREATEPATH);
```

При использовании gmJob::REMOTE, если путь *src* на удаленной системе не начинается с ‘/’, то он рассматривается относительно домашнего каталога пользователя. В именах исходных файлов/каталогов допускается задание масок “*” и “?”. *Внимание:* использование масок или флага gmShell::TEXT совместно с флагом gmJob::RECURSIVE поддерживается не всеми классами протокола доступа, при этом результат такого копирования может зависеть от типа локальной ОС и выбранного протокола. Для описания нескольких операций копирования, следует несколько раз вызвать функцию AddInFile. При этом файлы/каталоги будут копироваться в том же порядке, в котором были вызваны соответствующий функции AddInFile, за исключением элементов с флагом gmJob::REMOTE. Действие флага gmJob::CREATEPATH аналогично флагу gmShell::CREATEPATH в функции gmShell::StageIn().

```
void gmJob::AddOutFile(pCSTR dst, pCSTR workpath, unsigned flags = 0)
```

Функция определяет имя выходного файла или каталога во временной директории *workpath*, который при выполнении функции FetchResult (см. ниже) будет скопирован в файл/каталог *dst* на локальной или на удаленной системе. Параметры копирования задаются аналогично функции AddInFile, при этом наличие флага gmJob::CREATEPATH означает создание выходной директории *dst* на локальной или удаленной системе. Если аргумент *workpath* содержит строку “STDOUT” или “STDERR”, то в файл *dst* копируется вывод приложения в поток stdout/stderr. В этом случае *dst* должен быть именем файла, а не каталога. При копировании stdout/stderr флаг gmJob::TEXT устанавливается автоматически. Альтернативой использованию STDOUT/STDERR является перенаправление вывода в заданный пользователем файл в команде *command* (см. ниже).

```
void gmJob::ClearInFileList()
void gmJob::ClearOutFileList()
```
Очищает списки входных или выходных файлов, заданные функциями AddInFile и AddOutFile.

```
wxString gmJob::command
```
Строка, задающая набор команд пользователя, которые будут выполнены при запуске задания. Текущим каталогом при выполнении этой команды будет временный каталог для данного задания на удаленной системе. Таким образом, если запускаемый файл указан как входной файл с помощью AddInFile, он будет скопирован во временный каталог и может быть запущен командой "./<имя файла>" (в среде Unix). Если требуется указать несколько команд, их следует разделять символами ';' (в bash shell) или '\n' (в bash shell и batch-файлах). В некоторых случаях, например, при копировании файлов с Windows-машины, необходимо установить атрибут файла для запуска, т.е. строка *command* должна выглядеть так "chmod u+x <имя файла>; ./<имя файла>". Альтернативой является задание пути к постоянному файлу на удаленном ресурсе, например "/usr/bin/perl /my_dir/my_script.pl". В этом случае его не надо описывать как входной файл. Строка *command* не должна содержать одинарных или двойных кавычек. Если кавычки необходимы, то команду следует записать в текстовый файл (скрипт), загрузить его вместе с другими входными файлами, а в *command* поместить команду запуска этого скрипта.

```
bool gmJob::mpi
```
Если *mpi* = true, то данное задание будет быть запущено в среде MPI, т.е. содержимое *command* будет скопировано в отдельный скрипт и передано команде mpiexec (или аналогичной команде, в зависимости от настроек менеджера заданий).

```
unsigned gmJob::nproc, gmJob::ppn
```
Дополнительные параметры для MPI-задач: *nproc* – полное число процессов, *ppn* – число процессов на узел кластера. Если, например, *nproc* = 5, а *ppn* = 2, то будет запущено по два процесса на двух узлах, а также один процесс на третьем узле (всего 2*2+1=5 процессов). По умолчанию значения этих параметров равны нулю, что означает использование стандартных параметров для данной системы управления заданиями.

```
unsigned gmJob::walltime
```
Определяет максимальное время выполнения задания в секундах. По умолчанию *walltime* = 0, что означает использование стандартных параметров для данной системы управления заданиями.

```
wxString gmJob::forerunner
```
Если этот параметр содержит идентификатор другого задания <ref_job_id>, то команды пользователя из текущего задания будут выполнятся в рабочем каталоге <ref_job_id>. Задание <ref_job_id> должно быть сформировано на той же удаленной системе и на момент запуска Submit для текущего задания иметь статус >= JOB_PREPARED. Эта переменная применяется, как правило, чтобы избежать излишнего копирования данных при использовании выходных файлов одного задания в качестве входных файлов для другого.

```
unsigned gmJob::type
```
Определяет дополнительные свойства (тип) задания. Может принимать значения: gmJob::NORMAL (по умолчанию) и gmJob::DUMMY. Если задание имеет тип gmJob::DUMMY, то при выполнении функции Submit оно фактически не выполняется на уда-

ленной системе, а сразу переводится в состояние JOB_COMPLETED. Задания типа gmJob::DUMMY с установленным значением *forerunner* могут использоваться для выгрузки файлов из рабочих каталогов ранее запущенных задач.

2.7 Запуск и управление заданием

В этом разделе перечислены функции-члены класса gmJob, предназначенные для управления заданиями. В процессе выполнения задание проходит через последовательность состояний, описание которых приводится в описании функции GetState.

```
int gmJob::GetState()
```

Определение текущего состояния задания. Возвращаемые значения:

```
JOB_INIT          инициализация (состояние по умолчанию после создания объ-
                  екта gmJob);
JOB_PREPARED      задание подготовлено к выполнению: временный каталог соз-
                  дан, входные файлы в скопированы, осуществлена привязка к
                  менеджеру заданий;
JOB_SUBMITTED     задание передано системе управления заданиями;
JOB_QUEUED        задание поставлено в очередь;
JOB_RUNNING       задание выполняется;
JOB_SUSPENDED     задание приостановлено;
JOB_EXITING       выполнение окончено и происходит удаление из очереди;
JOB_COMPLETED     задание полностью выполнено и выходные данные доступны для
                  копирования;
JOB_HAVERESULT    выходные данные скопированы на локальную машину;
JOB_FAILED        при выполнении одной из операций произошла ошибка.
```

Следует иметь в виду, что если задание было поставлено в очередь или выполняется на удаленной машине, то вызов GetState() может приводить к некоторой задержке, связанной к обращению к удаленному ресурсу.

```
int gmJob::LastState()
```

Состояние задания на момент окончания последней операции. Данная функция возвращает значение аналогично GetState, но ее вызов не приводит к обращению к менеджеру заданий.

```
static pCSTR StateName(int state)
```

Возвращает текстовую строку, описывающую указанное состояние.

```
int gmJob::Submit(gmJobManager& mngr, pCSTR user_id=NULL, bool mktemp=true)
int gmJob::Submit(pCSTR user_id=NULL, bool mktemp=true)
```

Функция передает задание на выполнение созданному ранее менеджеру заданий *mngr*. Для управляемых объектов gmJob следует использовать второй вариант функции без аргумента *mngr*, т.к. эти задания привязываются к менеджеру уже на этапе их создания. Идентификатор задания при *mktemp* = true генерируется менеджером автоматически с учетом того, чтобы все задания имели уникальные идентификаторы (используется утилита Unix mktemp). При этом пользователь может задать начальную часть идентификатора в строке *user_id*. Если *user_id* = NULL, то начальная часть называется “void”. В том случае, когда *mktemp* = false, полный идентификатор задания определяется пользователем в непустой строке *user_id*. При ручном задании идентификатора пользователь должен обеспечить, чтобы все задания, запущенные на одном и том же удаленном ресурсе имели разные полные идентификаторы. Перед тем, как задание будет передано системе управления заданиями, на удаленном ресурсе будет создан временный каталог с именем, основанном на идентификаторе. В этот каталог будут скопированы входные файлы и каталоги, причем гарантируется, что после завершения Submit все операции копирования завершены. Функция возвращает управление, не дожидаясь окон-

чания выполнения задания. Дальнейшее состояние задания можно узнать с помощью функции GetState. При выполнении Submit ссылка *mngr* сохраняется в объекте класса gmJob, т.е. задание привязывается к указанному менеджеру. В случае ошибки постановки задания в очередь или ошибки копирования хотя бы одного из входных файлов заданию присваивается состояние JOB_FAILED. Если превышено ограничение на количество задач в очереди для данной системы очередей (или превышен предел, установленный параметром *jobs_limit*), задание переводится в состояние JOB_PREPARED. В дальнейшем оно будет запущено автоматически, когда менеджер определит, что одно из работающих заданий завершилось. В случае успешной постановки в очередь задание переходит в состояние JOB_SUBMITTED. Функция возвращает конечное состояние задания.

```
wxString gmJob::GetID()
```

Функция возвращает идентификатор задания, если оно было запущено, в ином случае возвращает пустую строку.

```
wxString gmJob::GetWorkDir()
```

Функция возвращает путь к рабочему каталогу на удаленной системе для данного задания, если оно было запущено, в ином случае возвращает пустую строку.

```
gmJob* gmJobManager::JobByID(pCSTR id) const
gmJob* gmJobManager::operator[](pCSTR id) const
```

Функция JobByID и переопределенный оператор [] позволяют найти задание по указанному идентификатору, например: `gmJob *myjob = manager["myjob"]`. Если задание не найдено, то возвращается NULL.

```
int gmJob::Wait()
```

Функция ожидает пока запущенное ранее задание не перейдет в состояние JOB_COMPLETED, JOB_FAILED или пока не истечет максимальное время ожидания, заданное в переменной класса gmJobManager::wait_timeout (в миллисекундах). Следует заметить, что копирование выходных файлов с удаленного ресурса не происходит автоматически, для этого служит функция FetchResult. Функция возвращает текущее состояние аналогично GetState() и генерирует исключение, если задание не было запущено.

```
int gmJob::FetchResult(bool fWait=true)
```

Копирование выходных файлов, заданны с помощью функций AddOutFile, из временного каталога на локальный компьютер и в постоянные каталоги на удаленном ресурсе (для файлов с флагом gmJob::REMOTE). Если в момент запуска FetchResult задание еще не завершено, то в зависимости от флага *fWait* функция либо ожидает окончания выполнения в течение времени gmJobManager::wait_timeout аналогично Wait (*fWait*=true), либо пытается скопировать текущую версию выходных файлов незавершенного задания (поддерживается не всеми менеджерами заданий). Если задание завершено (JOB_COMPLETED), то после успешного копирования выходных файлов FetchResult переводит его в состояние JOB_HAVERESULT, а при ошибке копирования в JOB_FAILED. При копировании выходных файлов незавершенного задания его состояние остается неизменным, несмотря на возможные ошибки копирования. Функцию FetchResult можно использовать также для заданий, находящихся в состоянии JOB_FAILED. Функция возвращает текущее состояние и генерирует исключение, если задание не было запущено.

```
int gmJob::StageOut(pCSTR dst, pCSTR workpath, unsigned flags = 0)
```

Копирование одного файла/каталога *workpath* из временного каталога в выходной файл/каталог *dst* на локальном компьютере или на удаленной системе (для файлов с флагом gmJob::REMOTE). Обычно, эта функция применяется для получения временного результата в процессе выполнения задания, если менеджер заданий предоставляет такую возможность. Функция возвращает 0 при успешном выполнении, ≠ 0 при ошибке копирования, и генерирует исключение, если задание не было запущено. Если аргумент *workpath* содержит строку “STDOUT” или “STDERR”, то в файл *dst* копируется вывод приложения в поток stdout/stderr. В этом случае *dst* должен быть именем файла, а не каталога. При копировании stdout/stderr флаг gmJob::TEXT устанавливается автоматически.

```
void gmJob::Stop()
```

Принудительная остановка задания. После выполнения этой функции задание всегда переходит в состояние JOB_FAILED. Функция генерирует исключение, если задание не было запущено.

```
void gmJob::Detach()
```

Происходит открепление объекта gmJob от менеджера, указанного при запуске задания, однако фактически задание продолжает выполняться. Управляемый объект gmJob уничтожается, а неуправляемый переводится в состояние JOB_INIT. В дальнейшем открепленный неуправляемый объект может быть уничтожен или использован для формирования другого задания, однако это никак не повлияет на выполнение того исходного задания, от которого он был откреплен. При уничтожении объекта gmJob функция Detach выполняется автоматически, если задание было связано с каким-либо менеджером.

```
void gmJob::Clear()
```

Происходит остановка задания, если оно было запущено, удаление временного каталога на удаленном ресурсе, открепление от менеджера заданий и очистка всех временных переменных. Управляемый объект gmJob уничтожается, а неуправляемый переводится в состояние JOB_INIT и, если необходимо, может быть использован для формирования другого задания и запуска его на любом менеджере заданий. При этом поля класса gmJob, устанавливаемые пользователем (*command, nproc, nthreads, walltime* и др.), а также списки входных и выходных файлов, не очищаются.

```
gmJob* gmJobManager::Restore(pCSTR id)
```

Менеджер производит поиск задания, запущенного менеджером того же класса на той же удаленной системе с идентификатором *id*, и, если задание найдено, создает управляемый объект gmJob. Эта функция обычно используется совместно с Detach для восстановления объектов gmJob по их идентификаторам. Объект может быть восстановлен, даже если указанное задание было создано другой программой. Функция возвращает адрес созданного динамического объекта или NULL, если задание с данным идентификатором не было найдено.

```
int gmJob::Attach(gmJobManager& mngr, pCSTR id)
```

Функция восстанавливает запущенное ранее задание аналогично Restore, при информация о найденном задании помещается в созданный пользователем неуправляемый объект gmJob. Функция возвращает текущее состояние задания или JOB_INIT, если задание с данным идентификатором не было найдено на удаленной системе. Если перед запуском Attach объект gmJob был связан с другим заданием, генерируется исключение.

2.8 Пакетная работа с заданиями

Функции, описанные в данном разделе, являются функциями-членами класса менеджера задания и предназначены для выполнения одинаковых действий над набором заданий, запущенных с помощью данного менеджера. Аргументом всех этих функций является строка *id_prefix*, которая указывает, что действия будут выполнены только для заданий, идентификаторы которых начинаются с данного набора символов. Если *id_prefix* = NULL, то действия выполняются над всеми заданиями. Как правило результат выполнения каждой пакетной функции аналогичен последовательному запуску аналогичной функции, описанной в предыдущем разделе, для каждого отдельного задания.

```
int gmJobManager::WaitAll(pCSTR id_prefix = NULL)
```

Ожидает окончания выполнения заданий и возвращает: JOB_COMPLETED – если все задания завершились успешно, JOB_FAILED – если хотя бы одно задание завершилось с ошибкой, JOB_INIT – если не найдено ни одного задания с данным префиксом *id_prefix*.

```
int gmJobManager::FetchAll(bool fWait=true, pCSTR id_prefix = NULL)
```

Копирует выходные файлы на локальную и удаленную системы и возвращает: JOB_HAVERESULT – если все данные были скопированы, JOB_FAILED – если хотя бы одно задание завершилось с ошибкой.

```
void gmJobManager::StopAll(pCSTR id_prefix = NULL)
```

Останавливает все запущенные задания.

```
void gmJobManager::ClearAll(pCSTR id_prefix = NULL)
```

Останавливает задания, удаляет их временные директории, сбрасывает состояния в JOB_INIT и убирает из списка заданий для данного менеджера.

```
void gmJobManager::DetachAll(pCSTR id_prefix = NULL)
```

Открепляет задания от менеджера, но не останавливает их. Эта функция автоматически выполняется при уничтожении объекта класса gmJobManager.

```
JobList gmJobManager::RestoreAll(pCSTR id_prefix = NULL)
```

Выполняет поиск всех заданий (аналогично Restore), запущенных ранее с помощью менеджера данного класса на той же удаленной системе и создает для них управляемые объекты gmJob. Функция не дублирует задания, которые уже прикреплены к данному менеджеру. Возвращаемое значение – объект STL list class (см. определение типа JobList в разделе 1.3), содержащий указатели на вновь созданные объекты gmJob.

```
JobList gmJobManager::GetJobList(pCSTR id_prefix = NULL) const
```

Возвращает объект STL list class, элементами которого являются указатели на все задания, прикрепленные к данному менеджеру. Т.к. возвращаемый список является копией, изменение самого списка не влияет на работу менеджера, однако действия над его элементами аналогичны действиям над соответствующими заданиями.

3 Загрузка параметров из XML-файла

В файле gridmd.h определен класс gmResourceDescr, который может использоваться для сохранения и загрузки параметров менеджеров заданий и протоколов доступа к удаленной системе в/из XML файла. Для инициализации объекта класса может использоваться конструктор или функция init:

```
gmResourceDescr(const int res_type_ = gmRES_DEFAULT_TYPE,
    const int shell_type_ = gmSHELL_DEFAULT_TYPE, int active_ = 1)

gmResourceDescr::init(const int res_type_ = gmRES_DEFAULT_TYPE,
    const int shell_type_ = gmSHELL_DEFAULT_TYPE, int active_ = 1)
```

Здесь параметр *res_type_* определяет тим менеджера заданий (см. таб. 4), а *shell_type_* – тип протокола доступа (см. таб. 5). По умолчанию при компиляции программы под Windows используется *res_type_* = gmRES_WINAPI, *shell_type_* = gmSHELL_WIN, при компиляции под Unix: *res_type_* = gmRES_BSH, *shell_type_* = gmSHELL_UNIX. Флаг *active_* говорит о том, является ли данный ресурс в настоящее время активным (применяется при описании нескольких ресурсов в менеджере сценариев gmManager).

Таблица 4. Выбор менеджера заданий в классе gmResourceDescr и XML-файле описания ресурсов.

Класс менеджера заданий	**Значение параметра res_type_ в gmResourceDescr::init**	**Значение атрибута type элемента <job_manager> в XML-файле**
gmPBSManager	gmRES_PBS	pbs
gmSLURMManager	gmRES_SLURM	slurm
gmBShManager	gmRES_BSH	bsh
gmWinShManager	gmRES_WSH	wsh
gmWinAPIManager	gmRES_WINAPI	win

Таблица 5. Выбор протокола доступа в классе gmResourceDescr и XML-файле описания ресурсов.

Класс протокола доступа	**Значение параметра shell_type_ в gmResourceDescr::init**	**Значение атрибута type в элементе <session> XML-файла**
gmShellPlink	gmSHELL_PLINK	plink
gmShellUnix	gmSHELL_UNIX	unix
gmShellLibssh	gmSHELL_LIBSSH	libssh
gmShellMinGW	gmSHELL_MINGW	mingw
gmShellWin	gmSHELL_WIN	cmd

Дополнительные параметры менеджеров заданий задаются с помощью хэш-таблицы

```
std::map<wxString,wxString> gmResourceDescr::param
```

в соответствии с таб. 3. Например:

```
cluster.init(gmRES_PBS, gmSHELL_PLINK, 1);
cluster.param["save_job_info"]   = "true";
cluster.param["jobs_limit"]      = "2";
cluster.param["qsub_args"]       = "-q mpi";
```

Параметры протокола доступа задаются аналогичном образом с помощью члена класса gmResourceDescr::shell в соответствии с таб. 1:

```
cluster.session.param["rem_tmp_dir"]    = "/home/username/.gmJobManager";
cluster.session.param["dump_commands"] = "true";
```

Параметры “host” и “login” могут задаваться как с помощью хэш-таблицы, так и через переменные-члены класса:

```
cluster.session.host  = "10.0.0.1";
cluster.session.login = "username";
```

В менеджере сценариев GridMD, описываемому объектом класса gmManager (подробное описание класса см. в полной документации к GridMD), совокупность данных о менеджере заданий, протоколе доступа и, возможно, запускаемом внешнем приложении называется ресурсом. Для добавления ресурса, описываемого объектом класса gmResourceDescr, применяется функция:

```
int gmManager::add_resource( const gmResourceDescr &rdescr,
                             const string &name)
```

где параметр *name* задает уникальное имя ресурса.

Сохранение/загрузка ресурсов в/из XML-файла производится функциями:

```
int gmManager::save_resources(const char *filename)
int gmManager::load_resources(const char *filename)
```

где *filename* – путь к XML-файлу.

Структура XML-файла имеет вид:

```
<?xml version="1.0" encoding="utf-8"?>
<flowgraph>
  <scheduler>
    <resource name="username@remote_host1" active="1">
      <session type="plink" host="10.0.0.1" login="username">
        <rem_perm_dir>/home/username/perm_dir</rem_perm_dir>
        <dump_commands>false</dump_commands>
        ...
      </session>
      <job_manager type="pbs">
        <save_job_info>true</save_job_info>
        <jobs_limit>2</jobs_limit>
        <qsub_args>-q mpi</qsub_args>
        ...
      </job_manager>
      <application name="workflow_skeletons.exe">
        <progdir>/home/username/bin/</progdir>
        <prefix>export LD_LIBRARY_PATH=/home/username/lib</prefix>
        <postfix>echo Finished</postfix>
      </application>
    </resource>
    <resource name=" username@remote_host2" active="0">
      ...
    </resource>
    ...
  </scheduler>
</flowgraph>
```

Описанию каждого ресурса соответствует элемент <*resource*>, который может включать по одному элементу <*session*>, <*job_manager*> и <*application*>. Атрибут *type* элемента <*session*> задает тип протокола доступа (см. таб. 5), а внутренние элементы <*session*> – необязательные параметры протокола в соответствии с таб. 1. Значения параметров “host” и “login” могут задаваться как в виде атрибутов <*session host*=”...”, *login*=”...”>, так и с помощью подэлементов <*host*> и <*login*>. Атрибуты элемента <*job_manager*> определяют менеджер за-

даний (см. таб. 4), а внутренние элементы <*job_manager*> – необязательные параметры в соответствии с таб. 3.