

HostProcControl: Руководство программиста

DocumentId:GradSoft-PR-r-22.08.2000-v1.0b

July 8, 2001

Contents

1	Назначение продукта	2
2	Общий порядок использования	2
2.1	Главный принцип	2
2.2	Краткое описание объектов	3
2.3	Авторизация доступа	4
2.4	Время жизни объектов	4
3	Описание объектов	5
3.1	Объект HostControlHome	5
3.1.1	Общие сведения	5
3.1.2	IDL-интерфейс	5
3.1.3	Описание метода HostControlHome::getHostControl	5
3.1.4	Пример	6
3.2	Объект HostControl	7
3.2.1	IDL-интерфейс	7
3.2.2	Описание методов	7
3.2.3	Примеры	11
3.3	Объект FileReader	11
3.3.1	IDL-интерфейс	12
3.3.2	Описание методов	12
3.3.3	Пример	13
3.4	Объект FileWriter	13
3.4.1	IDL-интерфейс	13
3.4.2	Описание методов	14
3.4.3	Пример	14
4	Описание исключений	15
4.1	InvalidLogin	15
4.2	AccessControlFailure	15
4.3	InvalidModeArgument	15
4.4	HostError	16

5	Пример клиентского приложения	16
6	Поддержка	17
7	История документа	18

1 Назначение продукта

HostProcControl - это CORBA сервис, предоставляющий API для удаленного управления операционной системой. Реализованные нами CORBA-объекты обеспечивают две группы вещей:

1. доступ к файловой системе удаленной машины в следующем объеме:
 - (a) чтение и запись файлов;
 - (b) удаление файлов;
 - (c) проверка статуса файлов;
2. доступ к процессам, выполняющимся на удаленной машине в следующем объеме:
 - (a) получение списка процессов;
 - (b) получение параметров выделенных процессов;
 - (c) запуск процессов;
 - (d) посылка сигналов.

Представленный нами пакет работает с разными моделями ORB и может быть использован на разных платформах (см. **Руководство Администратора** в файле `AdministrationGuide_rus.pdf` из комплекта поставки). Пакет написан на C++ и поставляется в исходных кодах.

2 Общий порядок использования

2.1 Главный принцип

Существует 4 типа объектов:

1. HostControlHome;
2. HostControl;
3. FileReader;
4. FileWriter.

Первый из них автоматически создается при запуске сервера, все остальные объекты создаются через него:

1. HostControl: создается объектом HostControlHome;
2. FileWriter и FileReader: создаются объектом HostControl.

В итоге, общий порядок доступа к функциональности HostProcControl выглядит так:

1. Инициализация ORB и получение инициальной ссылки сервиса;
2. Организация доступа к HostControlHome;
3. Создание объекта HostControl;
4. Использование объекта HostControl; создание, использование и уничтожение объектов FileWriter и FileReader;
5. Уничтожение объекта HostControl.

Порядок получения инициальной ссылки сервиса зависит от способа ее публикации, порядок публикации описан в **Руководстве Администратора** пакета.

2.2 Краткое описание объектов

1. Объект HostControlHome:
 - Автоматически создается при запуске сервера;
 - Осуществляет проверку прав доступа, по результатам которой создает либо не создает объект HostControl.
2. Объект HostControl:
 - Создается методом HostControlHome::getHostControl();
 - Предоставляет доступ к процессам, выполняющимся на удаленной машине; удаляет файлы, осуществляет проверку статуса файлов; создает объекты FileReader и FileWriter, предоставляющие API для чтения либо записи файлов;
 - Уничтожается методом HostControl::destroy().
3. Объект FileReader:
 - Создается методом HostControl::createFileReader();
 - Предоставляет API для чтения файла, расположенного на удаленной машине;
 - Уничтожается методом FileReader::close().
4. Объект FileWriter:
 - Создается методом HostControl::createFileWriter();
 - Предоставляет API для записи в файл, расположенный на удаленной машине;
 - Уничтожается методом FileWriter::close().

2.3 Авторизация доступа

Вся функциональность HostProcControl доступна через объект HostControl (работа выполняется либо собственными методами объекта HostControl, либо методами его дочерних объектов). Авторизация доступа осуществляется на этапе создания данного объекта. Для этого вызов метода HostControl.Home::getHostControl осуществляется с двумя параметрами, один из которых рассматривается как "имя" пользователя, другой - как его "пароль". В настоящий момент мы ограничиваемся тем, что считаем переданные значения единой записью ("ID клиента"), аналог которой должен быть найден в *списке пользователей* сервиса.

- Список пользователей представляет собой файл на сервере, состоящий из строк формата <name>:<password>, где <name> и <password> - подстроки; этот файл создается *перед* запуском сервера HostControlServer[.exe] и может редактироваться в процессе его работы (подробно см. **Руководство Администратора** пакета).
- Пример списка пользователей:

```
John : Johnson
      A.E.Belyaev:265-60-43
<another_name>:<another_password>
```

Если поиск оказывается удачным (ID клиента внесен в список пользователей) доступ считается разрешенным и объект HostControl создается, если нет - то нет.

2.4 Время жизни объектов

По умолчанию, создаваемые клиентом экземпляры HostControl, FileReader и FileWriter уничтожаются самим клиентом с помощью методов HostControl::destroy(), FileReader::close() и FileWriter::close() соответственно. Однако Вы можете потребовать, чтобы сервер автоматически уничтожал созданные Вами объекты в том случае, если они не использовались в течение заданного Вами интервала времени. Для этого, каждый объект имеет метод set_timeout(), который вызывается с параметром, определяющим число секунд, в течение которого объект может существовать и не использоваться. При этом следует помнить, что:

1. вызов set_timeout(0) означает "бесконечное время ожидания", т.е. контроль использования объекта отключается;
2. актом использования объекта считается
 - (a) создание объекта;
 - (b) вызов любого из методов объекта, за исключением методов close() и destroy();

3. объект `HostControl` не может быть автоматически уничтожен пока существует хотя бы один порожденный им объект `FileReader` или `FileWriter`;
4. фактическое уничтожение дочернего объекта `FileReader` /`FileWriter` всегда производится родительским объектом `HostControl` и также считается актом его "использования".

3 Описание объектов

3.1 Объект `HostControlHome`

3.1.1 Общие сведения

Объект `HostControlHome` автоматически создается при запуске сервера `HostControlServer[.exe]` и служит "корневым" объектом нашего сервиса. В зависимости от опций, с которыми был запущен сервер, доступ к нему можно получить тремя способами (подробно смотри в **Руководстве Администратора** пакета):

- С помощью `corbaloc` IOR в форме


```
HostControlService=corbaloc::<host>:<port>/HostControlService
```
- С помощью `NameService`, как объекту `HostControlService` в `RootNamingContext`
- С помощью объектной ссылки в строковом виде, которую сервер может записать в файл либо отобразить на стандартном устройстве вывода.

Назначение данного объекта состоит в том, чтобы проверять права доступа клиента, после чего создавать либо *не* создавать объект `HostControl` (см. раздел **Авторизация доступа 2.3**).

3.1.2 IDL-интерфейс

```
interface HostControlHome
{
    HostControl getHostControl(in string name, in string password)
                                raises(LoginIncorrect,AccessControlFailure);
};
```

3.1.3 Описание метода `HostControlHome::getHostControl`

1. **Назначение:** Метод `getHostControl` сравнивает фактические значения параметров `name` и `password`, рассматриваемых как комплексный "ID клиента", с полями записей, хранящихся в *списке пользователей* сервиса (см. раздел **Авторизация доступа 2.3**) Если переданная запись соответствует одной из существующих, создается объект `HostControl`. В противном случае генерируется исключение `LoginIncorrect 4.1`. В

случае "проверка невозможна" (список пользователей не найден либо не может быть открыт для чтения), генерируется исключение AccessControlFailure 4.2.

2. Параметры:

- (a) name = имя клиента
- (b) password = пароль клиента

3. Возвращаемое значение: объект HostControl

3.1.4 Пример

Пусть для у нас имеется инициальная ссылка в стиле corbaloc, переданная при помощи опции командной строки

```
-ORBInitRef HostControlService=corbaloc::<host>:<port>/HostControlService
```

Следующий участок кода получает доступ к объекту HostControlHome и осуществляет попытку создать объект HostControl:

```
/* необходимые переменные */
Object_var      obj;
HostControl_var hostControl;
HostControlHome_var hostControlHome;

/* инициализация ORB при помощи опций командной строки */
myORB = ORB_init(argc,argv);

/* получение доступа к корневому объекту сервиса */
obj = myORB->resolve_initial_references( "HostControlService" );
hostControlHome = HostControlHome::_narrow(obj);

/* выбор имени-пароля */
char name[]="name";
char password[]="password";

try {
    /* попытка создать объект HostControl */
    hostControl = hostControlHome->getHostControl(name,password);
} catch ( const LoginIncorrect& ex ) {
    /* здесь должен быть вывод предупреждения */
    goto quit;
}
/* здесь мы можем использовать созданный объект */
```

```
/* уничтожение объекта HostControl */
hostControl->destroy();

quit:
    myORB->destroy();
```

3.2 Объект HostControl

Объект HostControl является основным объектом нашего сервиса и осуществляет следующие вещи:

1. управление процессами;
2. удаление файлов, проверку статуса файлов;
3. создание вспомогательных объектов FileReader и FileWriter, обеспечивающих доступ к содержанию файлов.

3.2.1 IDL-интерфейс

```
interface HostControl
{
    ProcInfoSeq getProcsInfo() raises(HostError);
    ProcInfo getProcInfoByPid(in unsigned long pid) raises(HostError);

    unsigned long bornProcByName(in string name, in StringSeq args) raises(HostError);
    void killProcByName(in short signal, in string name) raises(HostError);
    void killProcByPid(in short signal, in unsigned long pid) raises(HostError);

    FileReader createFileReader(in string path) raises(HostError);
    FileWriter createFileWriter(in string path, in WriteMode mode) raises(HostError);

    boolean statFile(in string path, in char mode) raises(HostError);
    void rmFile(in string path) raises(HostError);

    void set_timeout(in unsigned long timeout);
    void destroy();
};
```

3.2.2 Описание методов

A. Средства управления процессами

- метод getProcInfoByPid

1. **Назначение:** Возвращает информацию о процессе, найденном по номеру.

2. **Параметры:**

(a) pid = номер процесса

3. **Возвращаемое значение:** Информация о процессе в структуре ProcInfo:

```
struct ProcInfo
{
    unsigned long pid;    // the process id
    string        name;  // the filename of the executable
    unsigned long ppid;  // the PID of the parent
};
```

• метод getProcsInfo

1. **Назначение:** Возвращает информацию о всех имеющихся в системе процессах;

2. **Параметры:** Нет

3. **Возвращаемое значение:** Информация о всех имеющихся в системе процессах в структуре ProcInfoSeq:

```
typedef sequence<ProcInfo> ProcInfoSeq;
```

• метод bornProcByName

1. **Назначение:** Запускает процесс; при успешном завершении возвращает номер созданного процесса.

2. **Параметры:**

(a) name = имя процесса

(b) args = список аргументов в виде последовательности строк StringSeq:

```
typedef sequence<string> StringSeq;
```

3. **Возвращаемое значение:** Нет

• метод killProcByName

1. **Назначение:** Уничтожает процесс, найденный по имени.

2. **Параметры:**

(a) signal = сигнал, который должен быть послан процессу

(b) name = имя процесса

3. **Возвращаемое значение:** Нет

• метод killProcByPid

1. **Назначение:** Уничтожает процесс, найденный по номеру.

2. Параметры:

- (a) signal = сигнал, который должен быть послан процессу
- (b) name = номер (идентификатор) процесса

3. Возвращаемое значение: Нет

Б. Средства доступа к файлам:

- метод statFile

1. **Назначение:** Проверяет одно из 8-ми (13-ти) стандартных предположений относительно статуса файла; возвращает true, если действительный статус файла соответствует заданному предположению и false в противном случае.

2. Параметры:

- (a) path = имя файла
- (b) mode = код предположения

3. **Возвращаемое значение:** true, если действительный статус файла соответствует заданному предположению, false в противном случае.

Связь между значениями mode и содержанием предположений определяется по следующей таблице:

Для всех платформ:

Значение mode	Предположение
'e'	файл существует
's'	файл существует и его размер не равен нулю
'c'	файл существует и является устройством
'd'	файл существует и является директорией
'f'	файл существует и не является директорией
'r'	файл существует и доступен для чтения
'w'	файл существует и доступен для записи
'x'	файл существует и доступен для запуска

Дополнительно для UNIX:

Значение mode	Предположение
'h'	файл существует и является символической связью
'b'	файл существует и является специальным блоком
'p'	файл существует и является именованным каналом
'u'	файл существует и может использовать функцию setuid
'g'	файл существует и может использовать функцию setgid

Прочие значения mode недопустимы: генерируется исключение InvalidModeArgument 4.3

- метод `rmFile`
 1. **Назначение:** Удаляет файл.
 2. **Параметры:**
 - (a) `path` = имя файла, который должен быть удален
 3. **Возвращаемое значение:** Нет
- метод `createFileReader`
 1. **Назначение:** Создает объект типа `FileReader`, предоставляющий API для чтения файла, расположенного на удаленной машине.
 2. **Параметры:**
 - (a) `path` - имя файла, который будет открыт для чтения
 3. **Возвращаемое значение:** Объект типа `FileReader`
- метод `createFileWriter`
 1. **Назначение:** Создает объект типа `FileWriter`, предоставляющий API для записи в файл, расположенный на удаленной машине.
 2. **Параметры:**
 - (a) `path` = имя файла, который будет открыт для записи;
 - (b) `mode` = режим записи: этот параметр имеет смысл, если названный файл уже существует:
 - (`mode == WriteMode::enRewrite`) - существующий файл затирается;
 - (`mode == WriteMode::enAppend`) - информация дописывается в конец существующего файла.

В `HostControl.idl` перечисление `WriteMode` определено так:

```
enum WriteMode { enRewrite, enAppend };
```
 3. **Возвращаемое значение:** Объект типа `FileWriter`

V. Служебные средства:

- метод `set_timeout`
 1. **Назначение:** Устанавливает время в секундах, в течение которого объект `HostControl` может существовать и не использоваться (см. раздел *Время жизни объектов 2.4*).
 2. **Параметры:** время ожидания в секундах
 3. **Возвращаемое значение:** Нет
- метод `destroy`
 1. **Назначение:** Уничтожает объект `HostControl`
 2. **Параметры:** Нет
 3. **Возвращаемое значение:** Нет

3.2.3 Примеры

Следующий фрагмент кода осуществляет вывод списка существующих в системе процессов на стандартное устройство вывода:

```
/* создаем объект HostControl */
try {
    hostControl = hostControlHome->getHostControl("user","test");
} catch ( const LoginIncorrect& ex ) {
    /* вывод предупреждения */
    goto quit;
}

ProcInfoSeq_var procseq;

/* получение списка процессов */
procseq = hostControl->getProcsInfo();

/* вывод списка процессов на экран */
for(int i = 0; i< procseq->length() ; i++)
{
    cout << procseq[i].pid << " "
         << procseq[i].name << " "
         << procseq[i].ppid << endl;
}

hostControl->destroy();

quit:
.....
```

Следующий фрагмент кода удаляет лишний файл garbage.txt:

```
char name[20]="user", password[20]="test";
HostControl_var hostControl = hostControlHome->getHostControl(name,password);
if ( hostControl->statFile("garbage.txt",'f') ) {
    hostControl->rmFile("garbage.txt")
}
hostControl->destroy();
```

3.3 Объект FileReader

Создается объектом HostControl. Предоставляет доступ к неинтерпретируемой последовательности байт из файла, открытого на этапе создания объекта

при помощи `HostControl::createFileReader` 3.2.2.

3.3.1 IDL-интерфейс

```
interface FileReader
{
    OctSeq read(in unsigned long nbytes, out unsigned long actual_nbytes) raises(HostError);

    boolean eof() raises(HostError);

    void set_timeout(in unsigned long timeout );

    void close() raises(HostError);
};
```

3.3.2 Описание методов

- метод `read`
 1. **Назначение:** Считывает данные из удаленного файла и передает их вызвавшей процедуре.
 2. **Параметры:**
 - (a) `nbytes` = количество байт, которые надо считать;
 - (b) `actual_nbytes` = количество байт, которые были считаны в действительности;
 3. **Возвращаемое значение:** Считанные данные.
- метод `eof`
 1. **Назначение:** возвращает `true`, если при чтении данных достигнут конец файла, и `false` в противном случае
 2. **Параметры:** нет
 3. **Возвращаемое значение:** `true`, если при чтении данных достигнут конец файла, `false` в противном случае
- метод `set_timeout`
 1. **Назначение:** Устанавливает время в секундах, в течение которого объект `FileReader` может существовать и не использоваться (см. раздел *Время жизни объектов* 2.4).
 2. **Параметры:** время ожидания в секундах
 3. **Возвращаемое значение:** Нет
- метод `close`

1. **Назначение:** Закрывает файл и уничтожает объект.
2. **Параметры:** Нет
3. **Возвращаемое значение:** Нет

3.3.3 Пример

Следующий фрагмент кода осуществляет вывод на экран удаленного текстового файла:

```
/* создать объект FileReader */
FileReader_var freader = hostControl->createFileReader("demo.txt");

OctSeq_var data = new OctSeq;
const int buffer_size = 10000;
data -> length( buffer_size );

char buff[buffer_size+1];
int readed;
do {
    /* считать порцию данных из файла */
    data = freader->read( buffer_size, readed );

    /* вывести данные на экран */
    memcpy(buff,data->get_buffer(),readed);
    buff[readed+1]='\0';
    cout << buff;
} while ( readed==buffer_size );

/* закрыть файл и уничтожить объект */
freader->close();
```

3.4 Объект FileWriter

Создается объектом HostControl. Записывает неинтерпретируемую последовательность байт в удаленный файл, открытый на этапе создания объекта при помощи HostControl::createFileWriter 3.2.2.

3.4.1 IDL-интерфейс

```
interface FileWriter
{
    void    write(in unsigned long nbytes, in OctSeq bytes) raises(HostError);

    void    set_timeout(in unsigned long timeout);

    void    close() raises(HostError); \
};
```

3.4.2 Описание методов

- метод write
 1. **Назначение:**
Записывает данные в файл.
 2. **Параметры:**
 - (a) nbytes : количество байт, которые должны быть записаны в файл;
 - (b) bytes : собственно данные, которые будут записаны;
 3. **Возвращаемое значение:** Нет
- метод set_timeout
 1. **Назначение:**
Устанавливает время в секундах, в течение которого объект FileWriter может существовать и не использоваться (см. раздел *Время жизни объектов* 2.4).
 2. **Параметры:** время ожидания в секундах
 3. **Возвращаемое значение:** Нет
- метод close
 1. **Назначение:** Закрывает файл и уничтожает объект.
 2. **Параметры:** Нет
 3. **Возвращаемое значение:** Нет

3.4.3 Пример

Следующий фрагмент кода копирует локальный файл на удаленный компьютер:

```
/* создать объект FileWriter */
FileWriter_var filewriter =
    hostControl->createFileWriter( "demo.txt", UAKG_enRewrite );

const int buffer_size = 10000;
OctSeq_var buf = new OctSeq;
buf -> length( buffer_size );

int readed;
while ( !feof( file ) ) {
    /* прочитать данные из локального файла */
    readed = fread(buf->data(),1,buffer_size,file);

    /* записать данные в удаленный файл */
```

```

        filewriter->write(readed,buf);

        if ( ferror( file )) {
            break;
        }
    }
    /* закрыть файлы и уничтожить объект FileWriter */
    fclose( file );
    filewriter->close();

```

4 Описание исключений

4.1 InvalidLogin

Исключение InvalidLogin генерируется при попытке создать объект HostControl без права на его создание (смотри раздел **Авторизация доступа** 2.3, а также описание метода HostControlHome::getHostControl 3.1.3).

- IDL-интерфейс:

```
exception InvalidLogin {};
```

4.2 AccessControlFailure

Исключение AccessControlFailure генерируется при попытке создания объекта HostControl в том случае, если контроль прав доступа к объекту технически невозможен (список пользователей сервиса не найден либо не может быть открыт для чтения; смотри описание метода HostControlHome::getHostControl 3.1.3).

- IDL-интерфейс:

```
exception AccessControlFailure {};
```

4.3 InvalidModeArgument

Исключение InvalidModeArgument генерируется при вызове метода HostControl::statFile в том случае, если второй параметр метода (смотри описание HostControl::statFile 3.2.2) имеет запрещенное значение.

- IDL-интерфейс:

```
exception InvalidModeArgument {};
```

4.4 HostError

Исключение "общего назначения", генерируется в случае системной ошибки на удаленной машине.

- IDL-интерфейс:

```
exception HostError
{
    long    os_errno;
    string  errstr;
    string  add_info;
};
```

где отдельные поля имеют следующий смысл:

- `os_errno`: номер системной ошибки в соответствии с POSIX
- `errstr`: стандартное описание системной ошибки
- `add_info`: зарезервировано для дополнительного сообщения, которое может быть предоставлено разработчиками HostProcControl

5 Пример клиентского приложения

Ниже приведен пример клиентского приложения, демонстрирующий доступ к функциональности сервера посредством `corbaloc url`. Конкретный способ соответствующей настройки сервера описан в документации к ORB.

```
using namespace CORBA;
static ORB_var myORB ;

#define NAME_LEN 20
#define PASSWD_LEN 20

int main(int argc, char** argv)
{
    char name[20], password[20];
    myORB = ORB_init(argc, argv);
    Object_var obj;
    int retval=1;
    try {
        obj=myORB->string_to_object(
            "corbaloc::x.internal.company.com:1000/HostControlService");
    } catch ( const SystemException& ex ) {
        cerr << argv[0] << ": Can\'t resolve HostControlService reference" << endl;
        goto quit;
    }
}
```

```

}
if ( is_nil(obj) ) {
    cerr << "Object is NULL" << endl;
    goto quit;
}
HostControlHome_var hstCntrHome = HostControlHome::_narrow(obj);
if ( is_nil(HstCntrHome) ) {
    cout << "Can\'t narrow object " << endl;
    goto quit;
}
cout << "Enter name :" << flush ;
cin.get_line(name,NAME_LEN);
cout << "Enter password :" << flush;
cin.get_line(password,PASSWD_LEN);
HostControl_var hostControl ;
try {
    hostControl = HstCntrHome->getHostControl(name,password);
} catch (const SystemException& ex ) {
    cerr << "System Exception" << endl;
    goto quit;
} catch (const IncorrectLogin& ex ) {
    cerr << "Login incorrect" << endl;
    goto quit;
}
cout << "login succesfull" << endl;
retval=0;
//
// do you work here.
//
hostControl->destroy();
quit:
myORB->destroy();
return retval;
}

```

6 Поддержка

Пакет HostProcControl разработан и поддерживается компанией GradSoft, домашняя страница GradSoft <http://www.gradsoft.com.ua/> ; Представленная версия продукта - **HostProcControl-1.0**

7 История документа

05.04.2001 - отражены изменения в в интерфейсных спецификациях пакета: в предыдущей версии продукта **HostProcControl 1.0.b2** исключение `AccessControlFailure` отсутствовало, исключение `InvalidLogin` генерировалось при неудачной попытке создания объекта `HostControl` независимо от причины этой неудачи

23.02.2001 - выпуск версии **HostProcControl 1.0.b2**

22.08.2000 - первая редакция