CORBA File Server Specification

CORBAFileServer 2.1.9

HighQSoft GmbH Andreas Hofmann 2015/10/07

Contents

1 Introduction **1**

2 Use Cases **3**

2.1 Save 3

2.2 Read 3

2.3 Delete 3

2.4 Rename 4

3 Server **5**

3.1 Filename Handler 5

3.2 Security Handler 6

3.3 Terminator 7

3.4 Registration Name 8

3.5 String from Command Line 9

3.6 Starting as MS-Windows System Service 9

3.7 Using SSL for secure socket-based Transfers 9

4 Interfaces **11**

5 Plugin Support **25**

5.1 Interface 25

5.2 Initialization of a plugin 25

5.3 Default Implementation 26

5.4 Default Client 26

6 Licensing **29**

7 Properties **31**

8 Examples **37**

9 Modification History 41

Chapter 1

Introduction

ASAM ODS Servers are able to adminstrate the access to external files. The datatype `DT_-EXTERNALREFERENCE` was defined for that issue. This datatype includes the URL location of the file, a MIME type string and a placeholder for detailed description. The file should be saved at a central system because every user should have access to it.

The ASAM ODS specification doesn't define how a client can access these files. It depends on the particular environment what kind of protocol should be used to get the files. The URL location may define the access protocol but the security information for that must be provided by the client. To be a generic client, the implementation must be able to handle different protocols with different security models.

Here are some examples to point out just some problems:

Secure File Transfer Protocol (sftp)

This protocol is just a representative type of protocol ensuring a secure file network access. The security is the problem in that context. The system administrator has to register all ASAM ODS client users. This inceases the adminstrativ effort.

Hyper Text Transfer Protocol (http)

This protocol is designed for a public file access. Security is not the first aspect for applications using that protocol. In parallel a secure variant exists, but the problem for that is the same as for sftp.

Network Mounted Devices

Beside the security problems this kind of file share has the problem, that the client defines the place in the local filesystem where the device is mounted. But the URL location of the ASAM ODS entry is not useable for that intension.

There is no system that solves the problems for an ASAM ODS client without additional effort for administration.

The issue of the CORBAFileServer is to close that gap and to optimize the effort. Of course the client must implement additional code to handle external references.

Additionally there may be problems that cannot be solved by the client implementation, caused by wrong handling of the files and the according database references.

- File was deleted, reference already exists

- References was cleaned, file exists and need disc resources

These kinds of problems exist latently amd can be solved by a periodical database content analysis.

Chapter 2

Use Cases

This chapter descibes the use-cases for the service offered by the CORBAFileServer.

Think about an application model that defines an external reference to save a documentation file containing the description of the test equipment. The Applicationelement derived from AoTestEquipment gets the attribute that has the datatype `DS_EXTERNALREFERENCE`.

2.1 Save

The user has written the document that describes the equipment that he wants to use for the next tests. The client GUI offers the possibility to save the document at the designated attribute.

The client software uses the CORBA naming service to get the reference of the CORBAFileServer. The access to the CORBA naming service should be known, because the ASAM ODS server is also registered at this service. The software uses an API method to send the ASAM ODS session for authentication. The CORBAFileServer uses the transfered ASAM ODS session for a security check. Is the session valid and was the check successful the access will be allowed. The client software is able to use the API to save/transfer the file at a server location. The server itself defines the URL necessary for the location entry of the attribute. The ASAM ODS security ensures that the client is allowed to write the attribute.

2.2 Read

The user wants to read a document that is referenced by the attribute.

The ASAM ODS security ensures that the client is allowed to read the attribute. The client software uses the CORBA naming service to get the reference of the CORBAFileServer. The access to the CORBA naming service should be known, because the ASAM ODS server is also registered at this service. The software uses an API method to send the ASAM ODS session for authentication. The client uses the API method to get the file from CORBAFileServer by the stored URL location. The client it self must handle the data stream (write it to file or redirect it directly to an application).

2.3 Delete

The users wants to delete the document referenced by the attribut.

The ASAM ODS security ensures that the client is allowed to delete the attribute. The client software uses the CORBA naming service to get the reference of the CORBAFileServer. The access to the CORBA naming service should be known, because the ASAM ODS server is also registered at this service. The software uses an API method to send the ASAM ODS session for authentication. The client uses the API method to order the deletion of the file at server location.

2.4 Rename

The users wants to rename the document referenced by the attribut.

The ASAM ODS security ensures that the client is allowed to write the attribute. The client software uses the CORBA naming service to get the reference of the CORBAFileServer. The access to the CORBA naming service should be known, because the ASAM ODS server is also registered at this service. The software uses an API method to send the ASAM ODS session for authentication. The client uses the API method to order the rename action of the file at server location. This use-case can also be used to realize a kind of waste basket to collect all unused files.

Chapter 3

Server

In addition to the requirment by the client server interface, there are three other interfaces to increase the flexibility of the server.

3.1 Filename Handler

The main issue of the server is the creation and the adminstration of the serverside folder structure. The simplest case is the handling of a filesystem, but it is possible to have other complecated systems to save the byte streams comming from the client. The client should not involved in the type of storage and it should be transparent for the user.

To fulfill the different requirements the server should define a interface to change the behavior. Different installations may have different issues.

A class that should handle the datastorage names must implement the defined interface `com.highqsoft.corbafileserver.FilenameHandlerIF`. At server startup time the parameter `CORBAFileServer.FilenameHandler` can be used to pass the classname to the server implementation. The desired class must be localizable by the classpath of the JAVA VM.

The server uses a default class that implements the following filename rules:

- The folder structure depends on the actual date.
- The root directory can be configured via the keyword `CORBAFileServer.RootDir`
- If the root directory is not specified the implementation uses the folder `${user.home}/CORBAFileServer`.
- The implementation tries to save the file to the subfolder `year/month/day`
- To avoid overwritting of files the implementation appends subfolders step by step `[/minutes][/seconds][/milliseconds]`.
- If the file already exists inside the millisecond folder, the implementation throws an exception.
- The implemenation generated a file URL `file://[hostname]/[full local path]`

If you look at this interface you can see that CORBAFileServer passes the ASAM ODS session object, the id of the application element and the id of the instance element to the method that solves that problem. By this parameters you are able to build variable models for the datastorage.

The FilenameHandler is also reponsable for the rename and delete task. The default implementation writes the removed files to a waste folder that can be specified via the property `CORBAFileServer.WasteDir`. The default value of this property is `getRootDirectory()+File.separator+"waste"`.

3.1.1 The ODS FilenameHandler

There is a filename handler which use the definitions of the ASAM ODS Specification 5.2.0. The finanamehandler is implemented in `com.highqsoft.corbafileserver.ODSFilenameHandler` This filename handler loads context variables FILE_MODE, `FILE_NOTATION`, `FILE_ROOT_EXTREF` and `FILE_SYMBOLS` to find the location of the files and returns the URL accoring the settings of the ODS server.

3.2 Security Handler

Another task is the security handling. Different use-cases are possible depending on the destination environment. That is the reason why the server defines an interface for that issue. It is defined in the class `com.highqsoft.corbafileserver.SecurityHandlerIF` and can be configured by the property keyword `CORBAFileServer.SecurityHandler`.

A default implementation also exists that is used when no other class is given. This implementation just checks the incoming session by getting its name.

The method that should implement the task gets the ASAM ODS session, the id of the application element and the id of the instanc element. By this parameters you are able to build variable models for the security handling.

There is a switch defined via the property keyword `CORBAFileServer.SecurityActive` to disable the security handler. If the property value assigns `false`, the handler will not be used during whole runtime. The security handler is active by default. This switch is changeable at startup time only.

3.2.1 Writing a Security Handler

The best way to start is to extend the default security handler. The class `com.highqsoft.corbafileserver.SecurityHandler` implements the `com.highqsoft.corbafileserver.SecurityHandlerIF`. It checks just on valid ASAM ODS sessions by calling the `getName()` method of the `AoSession`.

```
class MySecurityHandler extends com.highqsoft.corbafileserver.SecurityHandler {
   ...
}
```

If your implementation needs some parameters, it can use the `com.highqsoft.corbafileserver.StreamFactory` singelton class that provides access to the properties of the configuration of the CORBAFileServer.

```
StreamFactory factory = StreamFactory.getInstance();
String classname = factory.getProperty("MySecurityHandler.OutputFile",
                     new File(System.getProperty("user.dir"), "MySecurityHandler.log").toString());
```

These properties can be specified in the `CORBAFileServer.cfg` file in the `[PARAMETERS]` section:

```
[Parameters]
-CORBAFileServer.ServiceName         = anyName
-CORBAFileServer.NameServiceHost     = anyHost
-CORBAFileServer.NameServicePort     = anyPort
-CORBAFileServer.RootDir             = anyFolder
-MySecurityHandler.OutputFile        = c:\tmp\my.log
```

To implement your issues, just overwrite the method of interest. If you want to check the default behavior first, just call the method of the super class.

If the implementation is done, create a new jar file and add it to the `[CLASSPATH]` section of the configuration.

```
[CLASSPATH]
%CORBAFileServer_ROOT%\jar\CORBAFileServer.jar
%CORBAFileServer_ROOT%\jar\MySecurityHandler.jar
```

3.3 Terminator

This class is optional and will be used to terminate the transaction. In compare to the filename handler the class can use the ASAM ODS session, the id of the application element and the id of the instance element to do that job. Different ideas can be realized by this class.

The interface is defined in the class `com.highqsoft.corbafileserver.TerminatorIF` and the name of implementation can be pass to server via the property keyword `CORBAFileServer.Terminator`.

The class must implement the following method:

```
void terminateForInstance (AoSession aoSession,
                           String name,
                           T_LONGLONG aid,
                           T_LONGLONG iid,
                           String parameter) throws CORBAFileServerException;
```

aoSession

the ASAM ODS session.

name

the name of the file.

aid

the application element id.

iid

the instance element id.

parameter

the parameter string. The content depends on the server side terminate implementation.

A default implementation doesn't exist.

But there is an useful example defined in class `com.highqsoft.corbafileserver.TerminateProcess`. This class creates a process that is defined by property `CORBAFileServer.Terminator.Command` or, if not present, by the client parameter string.

The property `CORBAFileServer.Terminator.Command` allows to specify the command line string. The class uses the `java.text.MessageFormat.format()` class to format the command string. There are four possible parameters:

{0}

the name of the file.

{1}

the application element id.

{2}

the instance element id.

{3}

the parameter coming from client.

An example for a simple ant call may be:

```
ant -DCORBAFileServer.Filename="{0}" -F mybuild.xml
```

3.4 Registration Name

The server register itself at the CORBA naming service at startup time. The server must use a name and its CORBA reference for that issue.

The implementation handles different properties to control the name, that must be known at client side.

First of all there are the properties `CORBAFileServer.RegistrationFormat` and `CORBAFileServer.ServiceName`. The value of the registration format is `{0}.{1}@ {2}` by default and is passed to the static method `java.text.MessageFormat.format()` as shown in the following code snippet:

```
String format = ....getProperty("CORBAFileServer.RegistrationFormat", "{0}.{1}@{2}");
String serviceName = ....getProperty("CORBAFileServer.ServiceName");
String registerName = MessageFormat.format(format, new Object[] {
                            serviceName, "FileServer", getHostname()});
```

{0}

the service name.

{1}

the string value `FileServer`

{2}

the hostname.

Suppose the hostname of your server system is `sequoia`, your service name is `drain` and you are using the default behavior, then the registration name becomes:

```
drain.FileServer@sequoia
```

The server prints the registration name at startup time.

If you want to use the libraries on client side also, the hostname parameter will be the problem. The easiest way to do the right things is to pass the whole service name to the client and reduce the registration format to the first parameter.

```
CORBAFileServer.RegistrationFormat = {0}
CORBAFileServer.ServiceName = drain.FileServer@sequoia
```

By this way the client is able to use the same method to determine the service name.

3.5 String from Command Line

The server can be started from command:

```
$ java -jar CORBAFileServer.jar
```

The arguments are evaluated by the following rules:

- Arguments are handled as properties. That means each property is described by a keaword and a value.
- Each keyword has a leading minus '-' that will removed by the evaluation.
- The value of a property is separated by space and is located directly behind the keyword.
- It is allowed to omit the value for boolean value if the value should be `true`.

e.g.

```
$ java -jar CORBAFileServer.jar -CORBAFileServer.SecurityActive false
```

3.6 Starting as MS-Windows System Service

The CORBAFileServer is startable as system service for MS-Windows operating system. Use our **JVMService** tool to configure and install the service. The startup class `com.highqsoft.avalon.CORBAFileServerService` should be used for that issue. See the documentation of the **JVMService** for more information.

3.7 Using SSL for secure socket-based Transfers

The CORBAFileServer supports the encryption of socket-based transfers. This means that the methods

- getBySocket(...)
- getForInstanceBySocket(...)
- setBySocket(...)

- setForInstanceBySocket(...)

can transfer a data stream using the SSL facilities of Java. To do this, the CORBAFileServer supports a property called *CORBAFileServer.useSSL*. The value of this property is either **true** or **false**. If set to **true**, the CORBAFileServer will initiate SSLSockets and SSLServerSockets. In order to successfully use this feature, the administration of the CORBAFileServer has to provide valid key- and truststores to the Java VM. For more information regarding Java and SSL, see the documentation at oracle.com. In case the default key-/truststore of Java does not contain the keys/certificates used by the administration, the following startup arguments should be provided:

For the CFS client:

```
-Djavax.net.ssl.keyStore=/path/to/keystore
-Djavax.net.ssl.keyStorePassword=<keystore-password>
```

For the CFS:

```
-Djavax.net.ssl.trustStore=/path/to/truststore
-Djavax.net.ssl.trustStorePassword=<truststore-password>
```

Since the CFS client provides the ServerSocket, it needs a private key from the keystore to operate. The CFS itself uses the Socket and therefore needs a truststore with the public key of the ServerSocket.

The java tool **keytool** can be used to generate/manage the key-/truststores for the client and server. For more information on how to use keytool, please consult the tutorials at oracle.com.

Chapter 4

Interfaces

The CORBA IDL is located in the source code folder tree.

`etc/corbafileserver.idl`

The file `idlcompiler.bat` can be used to generated the JAVA code.

```
//****************************************************************************
// corbafileserver.idl
//
// This is the interface to the server.
//
// $Log: corbafileserver.idl,v $
// Revision 1.20  2014/12/04 10:30:26  alex
// Updated javadoc of close() and InputStreamIF
//
// Revision 1.19  2011/12/05 15:27:14  karst
// Add new error code.
//
// Revision 1.18  2011/12/05 14:41:49  alex
// Replaced typedef DS_LONG with LONG_ARRAY to avoid complications with ODS typedefs
//
// Revision 1.17  2011/12/02 15:07:58  alex
// Added new method CORBAFileServer.getSizes
// Increased INTFERFACEVERSION to 1.3
//
// Revision 1.16  2011/12/02 14:08:55  alex
// getVersion now reads the value from the IDL files
// instead from hardcoded java value
// Added new FileNameHandlerExt implementation that
// serves as wrapper for old FileNameHandlerIF
// implementations
// Modified CFS to cast FileNameHandlers to FileNameHandlerExtIF and wrap them if necessary
//
// Revision 1.15  2010/09/06 08:11:09  andy
// [getSize()#1393] Send a better exception when the URL doesn't exist or the file doesn't exist.
//
// Revision 1.14  2010/03/31 12:06:57  elke
// added note about keeping string version up to date
//
// Revision 1.13  2010/03/31 10:02:51  elke
// new functions: getSize, getSizeForInstance and getInterfaceVersion
//
// Revision 1.12  2010/03/12 14:58:16  andy
// Changes made by Reiner Knbl (EPOS/CAT) for the MDM/IDL3 compatibility.
//
// Revision 1.12  2010/03/12 15:27:00  knoebl
// Use the ODS 5.20 IDL
// Renamed paramters named "port" to "aPort" to avoid the IDL3 reserved word "port".
//
// Revision 1.11  2009/12/17 12:21:20  karst
// Use the ODS 5.2 IDL.
//
// Revision 1.10  2008/08/27 07:06:41  andy
// Quote system call.
//
// Revision 1.9  2007/05/24 12:12:06  andy
// Translate the documentation.
// Add more details about the registration format.
//
// Revision 1.8  2006/11/24 09:38:37  andy
// Termnate method added.
//
// Revision 1.7  2006/11/17 11:33:56  andy
// Enable the client to set the destination subfolder.
//
// Revision 1.6  2006/11/16 14:17:15  andy
// Socket stuff added.
//
// Revision 1.5  2006/11/14 10:33:39  andy
// Add the method reset() and length() to the input stream.
//
// Revision 1.4  2006/10/04 11:07:14  andy
// Move functionality added.
//
// Revision 1.3  2006/07/14 11:09:56  andy
// Add an exception and regenerated the code.
//
// Revision 1.2  2006/07/14 10:54:59  andy
// Methods added to the idl and newly generated.
//
// Revision 1.1  2006/01/20 14:32:17  andy
// Initial CVS-revision.
//
//*****************************************************************************


//*****************************************************************************
//*****************************************************************************
//
//    Please Note:
//
//    If functionality of this module is added/removed/changed
//    please increase the version number in String 'version'
//    of module Corbafileserver.java:
//           public static final String version = "1.0";
//    The function getInterfaceVersion returns that string!
//
//*****************************************************************************
//*****************************************************************************

#include "ods520.idl"

module com {
module highqsoft {
module corbafileserver {
module generated {

   typedef sequence<octet> DS_BYTE;
   typedef sequence<string> DS_STRING;
   typedef sequence<long long> LONG_ARRAY;

   /**
   * The error code.
```

```
*/
enum ErrorCode {
   FILESERVER_ASAMODS_EXCEPTION,
   FILESERVER_ACCESS_DENIED,
   FILESERVER_FILE_NOT_FOUND,
   FILESERVER_IO_EXCEPTION,
   FILESERVER_INFORMATION,
   FILESERVER_BAD_PARAMETER,
   FILESERVER_MISSING_PARAMETER,
   FILESERVER_CONNECT_FAILED,
   FILESERVER_CONNECT_REFUSED,
   FILESERVER_CONNECTION_LOST,
   FILESERVER_IMPLEMENTATION_PROBLEM,
   FILESERVER_NOT_IMPLEMENTED,
   FILESERVER_NO_MEMORY,
   FILESERVER_NULL_PARAMETER,
   FILESERVER_NOT_FOUND
};

/**
* The error severity flags.
*/
enum SeverityFlag {
   SUCCESS,     // Ok.
   INFORMATION, // Information.
   WARNING,     // Warning.
   ERROR        // Error.
};

/**
* The CORBAFileServer exception structure.
*/
exception CORBAFileServerException {
   ErrorCode    errCode;
   SeverityFlag sevFlag;
   string       reason;
};

/**
* The InputStreamIF is used to simulate the behavior of a Java InputStream over CORBA.
* The interface provides a method to read a certain amount of bytes from the
* InputStreamIF. The interface provides methods to get information about the stream as
* well as close it.
* <br/>
* Closing the stream is very important! The method must clean all server-side resource.
* This includes the stream itself if it is kept in the POA's active object map.
*/
interface InputStreamIF {

   /**
   * Reads up to len bytes of data from the input stream into an array
   * of bytes. An attempt is made to read as many as len bytes, but a
   * smaller number may be read, possibly zero. The number of bytes
   * actually read is returned as an integer.
   *
   * This method blocks until input data is available, end of file
   * is detected, or an exception is thrown.
   *
   * If b is null, a NullPointerException is thrown.
   *
   * If off is negative, or len is negative, or off+len is greater
   * than the length of the array b, then an IndexOutOfBoundsException is thrown.
   *
   * If len is zero, then no bytes are read and 0 is returned;
   * otherwise, there is an attempt to read at least one byte.
   * If no byte is available because the stream is at end of file,
   * the value -1 is returned; otherwise, at least one byte is read and stored into b.
```

```
     *
     * The first byte read is stored into element b[off], the next one into b[off+1],
     * and so on. The number of bytes read is, at most, equal to len. Let k be the number
     * of bytes actually read; these bytes will be stored in elements b[off] through b[off+k-1],
     * leaving elements b[off+k] through b[off+len-1] unaffected.
     *
     * In every case, elements b[0] through b[off] and elements b[off+len]
     * through b[b.length-1] are unaffected.
     *
     * If the first byte cannot be read for any reason other than end of file,
     * then an IOException is thrown. In particular, an IOException is thrown
     * if the input stream has been closed.
     *
     * The read(b, off, len) method for class InputStream simply calls the method
     * read() repeatedly. If the first such call results in an IOException,
     * that exception is returned from the call to the read(b, off, len) method.
     * If any subsequent call to read() results in a IOException, the exception
     * is caught and treated as if it were end of file; the bytes read up to that
     * point are stored into b and the number of bytes read before the exception
     * occurred is returned. Subclasses are encouraged to provide a more efficient
     * implementation of this method.
     *
     * @param b - the buffer into which the data is read.
     * @param off - the start offset in array b  at which the data is written.
     * @param len - the maximum number of bytes to read.
     * @return the total number of bytes read into the buffer,
     *         or -1 is there is no more data because the end
     *         of the stream has been reached.
     * @throws CORBAFileServerException if an IO exception occurs.
     */
    long read(out DS_BYTE b,
              in long off,
              in long len)
       raises (CORBAFileServerException);

    /**
     * Close the input stream. This method also can remove the object from the
     * server's active object map. <b>After this method is called, any other invocation
     * to a method of its object WILL lead to a CORBAException!</b>
     *
     * @throws CORBAFileServerException if an IO exception occurs.
     */
    void close()
       raises (CORBAFileServerException);

    /**
     * Get the length of the input stream.
     *
     * @return the length of the file to be transferd.
     * @throws CORBAFileServerException if an IO exception occurs.
     */
    long length()
       raises (CORBAFileServerException);

    /**
     * Reset the stream
     *
     * @throws CORBAFileServerException if an IO exception occurs.
     */
    void reset()
       raises (CORBAFileServerException);
 };

 interface CORBAFileServerIF {

    /*
* The version number of the IDL
```

```
*/
    const string INTERFACEVERSION = "1.3";


    /**
    * Save the data associated with the given intput stream.
    *
    * @throws CORBAFileServerException
    * with the following possible error codes:
    *    FILESERVER_CONNECT_FAILED
    *    FILESERVER_BAD_PARAMETER
    *    FILESERVER_CONNECTION_LOST
    *    FILESERVER_IMPLEMENTATION_PROBLEM
    *    FILESERVER_NOT_IMPLEMENTED
    *    FILESERVER_NO_MEMORY
    *
    * @param  aoSess the ASAM ODS session.
    * @param  name the name of the file.
    * @param  subDir an alternative sub directory, that can be specified,
    *                if the filename should not used to determine the destination folder.
    * @param  stream the input stream, ready to read by the server.
    * @return the url string of the created file.
    */
    string save (
       in org::asam::ods::AoSession aoSess,
       in string name,
       in string subDir,
       in InputStreamIF stream)
       raises (CORBAFileServerException);

    /**
    * Save the data associated with the given intput stream.
    * Specify the ApplicationElement id and the InstanceElement id
    * of the component that holds the external reference.
    *
    * @throws CORBAFileServerException
    * with the following possible error codes:
    *    FILESERVER_CONNECT_FAILED
    *    FILESERVER_BAD_PARAMETER
    *    FILESERVER_CONNECTION_LOST
    *    FILESERVER_IMPLEMENTATION_PROBLEM
    *    FILESERVER_NOT_IMPLEMENTED
    *    FILESERVER_NO_MEMORY
    *
    * @param  aoSess the ASAM ODS session.
    * @param  name the name of the file.
    * @param  subDir an alternative sub directory, that can be specified,
    *                if the filename should not used to determine the destination folder.
    * @param  aid the application element id.
    * @param  iid the instance element id.
    * @param  stream the input stream, ready to read by the server.
    * @return the url string of the created file.
    */
    string saveForInstance (
       in org::asam::ods::AoSession aoSess,
       in string name,
       in string subDir,
       in org::asam::ods::T_LONGLONG aid,
       in org::asam::ods::T_LONGLONG iid,
       in InputStreamIF stream)
       raises (CORBAFileServerException);

    /**
    * Save the data associated with the given intput stream.
    * Specify the name of an applciation element and the name of the instance element
    * that holds the external reference
    *
```

```
* @throws CORBAFileServerException
* with the following possible error codes:
*    FILESERVER_CONNECT_FAILED
*    FILESERVER_BAD_PARAMETER
*    FILESERVER_CONNECTION_LOST
*    FILESERVER_IMPLEMENTATION_PROBLEM
*    FILESERVER_NOT_IMPLEMENTED
*    FILESERVER_NO_MEMORY
*
* @param  aoSess the ASAM ODS session.
* @param  name the name of the file.
* @param  subDir an alternative sub directory, that can be specified,
*                if the filename should not used to determine the destination folder.
* @param  aeName the application element name.
* @param  ieName the instance element name.
* @param  stream the input stream, ready to read by the server.
* @return the url string of the created file.
*/
string saveForInstanceName (
   in org::asam::ods::AoSession aoSess,
   in string name,
   in string subDir,
   in string aeName,
   in string ieName,
   in InputStreamIF stream)
   raises (CORBAFileServerException);

/**
* Delete the data associated with the given name.
*
* @throws CORBAFileServerException
* with the following possible error codes:
*    FILESERVER_CONNECT_FAILED
*    FILESERVER_BAD_PARAMETER
*    FILESERVER_CONNECTION_LOST
*    FILESERVER_IMPLEMENTATION_PROBLEM
*    FILESERVER_NOT_IMPLEMENTED
*    FILESERVER_NO_MEMORY
*
* @param  aoSess the ASAM ODS session.
* @param  name the name of the file.
*/
void delete (
   in org::asam::ods::AoSession aoSess,
   in string name)
   raises (CORBAFileServerException);

/**
* Move the data associated with the given name.
*
* @throws CORBAFileServerException
* with the following possible error codes:
*    FILESERVER_CONNECT_FAILED
*    FILESERVER_BAD_PARAMETER
*    FILESERVER_CONNECTION_LOST
*    FILESERVER_IMPLEMENTATION_PROBLEM
*    FILESERVER_NOT_IMPLEMENTED
*    FILESERVER_NO_MEMORY
*
* @param  aoSess the ASAM ODS session.
* @param  urlo the url of the file.
*/
void move (
   in org::asam::ods::AoSession aoSess,
   in string url)
   raises (CORBAFileServerException);
```

```
/**
* Delete the data associated with the given name.
* Specify the ApplicationElement id and the InstanceElement id
* of the component that holds the external reference.
*
* @throws CORBAFileServerException
* with the following possible error codes:
*    FILESERVER_CONNECT_FAILED
*    FILESERVER_BAD_PARAMETER
*    FILESERVER_CONNECTION_LOST
*    FILESERVER_IMPLEMENTATION_PROBLEM
*    FILESERVER_NOT_IMPLEMENTED
*    FILESERVER_NO_MEMORY
*
* @param  aoSess the ASAM ODS session.
* @param  url the url of the file.
* @param  aid the application element id.
* @param  iid the instance element id.
*/
void deleteForInstance (
   in org::asam::ods::AoSession aoSess,
   in string url,
   in org::asam::ods::T_LONGLONG aid,
   in org::asam::ods::T_LONGLONG iid)
   raises (CORBAFileServerException);

/**
* Move the data associated with the given name.
* Specify the ApplicationElement id and the InstanceElement id
* of the component that holds the external reference.
*
* @throws CORBAFileServerException
* with the following possible error codes:
*    FILESERVER_CONNECT_FAILED
*    FILESERVER_BAD_PARAMETER
*    FILESERVER_CONNECTION_LOST
*    FILESERVER_IMPLEMENTATION_PROBLEM
*    FILESERVER_NOT_IMPLEMENTED
*    FILESERVER_NO_MEMORY
*
* @param  aoSess the ASAM ODS session.
* @param  name the name of the file.
* @param  aid the application element id.
* @param  iid the instance element id.
*/
void moveForInstance (
   in org::asam::ods::AoSession aoSess,
   in string name,
   in org::asam::ods::T_LONGLONG aid,
   in org::asam::ods::T_LONGLONG iid)
   raises (CORBAFileServerException);


/**
 * Get the data using a socket.
 *
 * @throws CORBAFileServerException
 * with the following possible error codes:
 *    FILESERVER_CONNECT_FAILED
 *    FILESERVER_BAD_PARAMETER
 *    FILESERVER_CONNECTION_LOST
 *    FILESERVER_IMPLEMENTATION_PROBLEM
 *    FILESERVER_NOT_IMPLEMENTED
 *    FILESERVER_NO_MEMORY
 *
 * @param  aoSess the ASAM ODS session.
```

```
     * @param  name the name of the file.
     * @param  host the hostname for the socket connection.
     * @param  port the port for the socket connection.
     */
       void getBySocket(
               in org::asam::ods::AoSession aoSess,
               in string name,
               in string host,
               in long aPort)
               raises (CORBAFileServerException);

    /**
     * Get the data using a socket.
     *
     * @throws CORBAFileServerException
     * with the following possible error codes:
     *    FILESERVER_CONNECT_FAILED
     *    FILESERVER_BAD_PARAMETER
     *    FILESERVER_CONNECTION_LOST
     *    FILESERVER_IMPLEMENTATION_PROBLEM
     *    FILESERVER_NOT_IMPLEMENTED
     *    FILESERVER_NO_MEMORY
     *
     * @param  aoSess the ASAM ODS session.
     * @param  name the url specification of the file.
     * @param  aid the application element id.
     * @param  iid the instance element id.
     * @param  host the hostname for the socket connection.
     * @param  port the port for the socket connection.
     */
    void getForInstanceBySocket(
                   in org::asam::ods::AoSession aoSess,
                   in string name,
                   in org::asam::ods::T_LONGLONG aid,
                   in org::asam::ods::T_LONGLONG iid,
                   in string host,
                   in long aPort)
                   raises (CORBAFileServerException);

    /**
     * Save the data using a socket.
     *
     * @throws CORBAFileServerException
     * with the following possible error codes:
     *    FILESERVER_CONNECT_FAILED
     *    FILESERVER_BAD_PARAMETER
     *    FILESERVER_CONNECTION_LOST
     *    FILESERVER_IMPLEMENTATION_PROBLEM
     *    FILESERVER_NOT_IMPLEMENTED
     *    FILESERVER_NO_MEMORY
     *
     * @param  aoSess the ASAM ODS session.
     * @param  name the name of the file.
     * @param  subDir an alternative sub directory, that can be specified,
     *                if the filename should not used to determine the destination folder.
     * @param  host the hostname for the socket connection.
     * @param  port the port for the socket connection.
     * @return the url string of the created file.
     */
    string saveBySocket(
             in org::asam::ods::AoSession aoSess,
             in string name,
             in string subDir,
             in string host,
             in long aPort)
             raises (CORBAFileServerException);
```

```
/**
 * Save the data using a socket.
 *
 * @throws CORBAFileServerException
 * with the following possible error codes:
 *    FILESERVER_CONNECT_FAILED
 *    FILESERVER_BAD_PARAMETER
 *    FILESERVER_CONNECTION_LOST
 *    FILESERVER_IMPLEMENTATION_PROBLEM
 *    FILESERVER_NOT_IMPLEMENTED
 *    FILESERVER_NO_MEMORY
 *
 * @param  aoSess the ASAM ODS session.
 * @param  name the name of the file.
 * @param  subDir an alternative sub directory, that can be specified,
 *                if the filename should not used to determine the destination folder.
 * @param  aid the application element id.
 * @param  iid the instance element id.
 * @param  host the hostname for the socket connection.
 * @param  port the port for the socket connection.
 * @return the url string of the created file.
 */
string saveForInstanceBySocket(
            in org::asam::ods::AoSession aoSess,
            in string name,
            in string subDir,
            in org::asam::ods::T_LONGLONG aid,
            in org::asam::ods::T_LONGLONG iid,
            in string host,
            in long aPort)
            raises (CORBAFileServerException);

/**
 * Save the data using a socket.
 *
 * @throws CORBAFileServerException
 * with the following possible error codes:
 *    FILESERVER_CONNECT_FAILED
 *    FILESERVER_BAD_PARAMETER
 *    FILESERVER_CONNECTION_LOST
 *    FILESERVER_IMPLEMENTATION_PROBLEM
 *    FILESERVER_NOT_IMPLEMENTED
 *    FILESERVER_NO_MEMORY
 *
 * @param  aoSess the ASAM ODS session.
 * @param  name the name of the file.
 * @param  subDir an alternative sub directory, that can be specified,
 *                if the filename should not used to determine the destination folder.
 * @param  aeName the application element name.
 * @param  ieName the instance element name.
 * @param  host the hostname for the socket connection.
 * @param  port the port for the socket connection.
 * @return the url string of the created file.
 */
string saveForInstanceNameBySocket(
         in org::asam::ods::AoSession aoSess,
         in string name,
         in string subDir,
         in string aeName,
         in string ieName,
         in string host,
         in long aPort)
         raises (CORBAFileServerException);

/**
* Read the data associated with the given name.
*
```

```
* @throws CORBAFileServerException
* with the following possible error codes:
*    FILESERVER_CONNECT_FAILED
*    FILESERVER_BAD_PARAMETER
*    FILESERVER_CONNECTION_LOST
*    FILESERVER_IMPLEMENTATION_PROBLEM
*    FILESERVER_NOT_IMPLEMENTED
*    FILESERVER_NO_MEMORY
*
* @param  aoSess the ASAM ODS session.
* @param  name the name of the file.
* @param  stream the input stream, ready to read by the server.
*/
InputStreamIF read (
   in org::asam::ods::AoSession aoSess,
   in string name)
   raises (CORBAFileServerException);

/**
* Read the data associated with the given name.
*
* @throws CORBAFileServerException
* with the following possible error codes:
*    FILESERVER_CONNECT_FAILED
*    FILESERVER_BAD_PARAMETER
*    FILESERVER_CONNECTION_LOST
*    FILESERVER_IMPLEMENTATION_PROBLEM
*    FILESERVER_NOT_IMPLEMENTED
*    FILESERVER_NO_MEMORY
*
* @param  aoSess the ASAM ODS session.
* @param  name the name of the file.
* @param  aid the application element id.
* @param  iid the instance element id.
*/
InputStreamIF readForInstance (
   in org::asam::ods::AoSession aoSess,
   in string name,
   in org::asam::ods::T_LONGLONG aid,
   in org::asam::ods::T_LONGLONG iid)
   raises (CORBAFileServerException);

/**
*  Get size of the file associated with the given name.
*
* @throws CORBAFileServerException
* with the following possible error codes:
*    FILESERVER_CONNECT_FAILED
*    FILESERVER_BAD_PARAMETER
*    FILESERVER_CONNECTION_LOST
*    FILESERVER_IMPLEMENTATION_PROBLEM
*    FILESERVER_NOT_IMPLEMENTED
*    FILESERVER_NO_MEMORY
*
* @param  aoSess the ASAM ODS session.
* @param  name the name of the file.
* @return the size of the input stream.
*/
long long getSize (
   in org::asam::ods::AoSession aoSess,
   in string name)
   raises (CORBAFileServerException);

/**
*  Get size of the file associated with the given name.
*
* @throws CORBAFileServerException
* with the following possible error codes:
*    FILESERVER_CONNECT_FAILED
*    FILESERVER_BAD_PARAMETER
*    FILESERVER_CONNECTION_LOST
*    FILESERVER_IMPLEMENTATION_PROBLEM
*    FILESERVER_NOT_IMPLEMENTED
*    FILESERVER_NO_MEMORY
*
* @param  aoSess the ASAM ODS session.
* @param  name the name of the file.
* @param  aid the application element id.
* @param  iid the instance element id.
* @param  size the size of the input stream.
*/
long long getSizeForInstance (
   in org::asam::ods::AoSession aoSess,
   in string name,
   in org::asam::ods::T_LONGLONG aid,
   in org::asam::ods::T_LONGLONG iid)
   raises (CORBAFileServerException);

/**
* This method can be called by the client when the server should be start a termination process.
*
* @throws CORBAFileServerException
* with the following possible error codes:
*    FILESERVER_CONNECT_FAILED
*    FILESERVER_BAD_PARAMETER
*    FILESERVER_CONNECTION_LOST
*    FILESERVER_IMPLEMENTATION_PROBLEM
*    FILESERVER_NOT_IMPLEMENTED
*    FILESERVER_NO_MEMORY
*
* @param  aoSess the ASAM ODS session.
* @param  name the name of the file.
* @param  parameter the parameter string. The content depends on the
*         server side terminate implementation.
*/
void terminate (
   in org::asam::ods::AoSession aoSess,
   in string name,
   in string parameter)
   raises (CORBAFileServerException);

/**
* This method can be called by the client when the server should be start a termination process.
*
* @throws CORBAFileServerException
* with the following possible error codes:
*    FILESERVER_CONNECT_FAILED
*    FILESERVER_BAD_PARAMETER
*    FILESERVER_CONNECTION_LOST
*    FILESERVER_IMPLEMENTATION_PROBLEM
*    FILESERVER_NOT_IMPLEMENTED
*    FILESERVER_NO_MEMORY
*
* @param  aoSess the ASAM ODS session.
* @param  name the name of the file.
* @param  aid the application element id.
* @param  iid the instance element id.
* @param  parameter the parameter string. The content depends on the
*         server side terminate implementation.
*/
void terminateForInstance (
   in org::asam::ods::AoSession aoSess,
   in string name,
   in org::asam::ods::T_LONGLONG aid,
      in org::asam::ods::T_LONGLONG iid,
      in string parameter)
      raises (CORBAFileServerException);

   /**
   * Get the name of the host where the server is running
   *
   * @param  aoSess the ASAM ODS session.
   * @return the hostname
   * @throws CORBAFileServerException
   * with the following possible error codes:
   *    FILESERVER_CONNECTION_LOST
   *    FILESERVER_IMPLEMENTATION_PROBLEM
   *    FILESERVER_NOT_IMPLEMENTED
   *    FILESERVER_NO_MEMORY
   */
   string getHostname (
      in org::asam::ods::AoSession aoSess)
      raises (CORBAFileServerException);

   /**
   * Get a context variable.
   *
   * @param  aoSess the ASAM ODS session.
   * @param key the keyword of the context value.
   * @return the context value
   * @throws CORBAFileServerException
   * with the following possible error codes:
   *    FILESERVER_CONNECTION_LOST
   *    FILESERVER_IMPLEMENTATION_PROBLEM
   *    FILESERVER_NOT_IMPLEMENTED
   *    FILESERVER_NO_MEMORY,
   *    FILESERVER_NOT_FOUND
   */
   string getContext(
      in org::asam::ods::AoSession aoSess,
      in string key)
      raises (CORBAFileServerException);

   /**
   * Set a context variable.
   *
   * @param  aoSess the ASAM ODS session.
   * @param key the keyword of the context value.
   * @param value the context value.
   * @throws CORBAFileServerException
   * with the following possible error codes:
   *    FILESERVER_CONNECTION_LOST
   *    FILESERVER_IMPLEMENTATION_PROBLEM
   *    FILESERVER_NOT_IMPLEMENTED
   *    FILESERVER_NO_MEMORY
   */
   void setContext(
      in org::asam::ods::AoSession aoSess,
      in string key,
      in string value)
      raises (CORBAFileServerException);

   /**
   * Remove a context variable.
   *
   * @param  aoSess the ASAM ODS session.
   * @param key the keyword of the context value.
   * @throws CORBAFileServerException
   * with the following possible error codes:
   *    FILESERVER_CONNECTION_LOST
   *    FILESERVER_IMPLEMENTATION_PROBLEM
```

```
     *    FILESERVER_NOT_IMPLEMENTED
     *    FILESERVER_NO_MEMORY
     */
     void removeContext(
        in org::asam::ods::AoSession aoSess,
        in string key)
        raises (CORBAFileServerException);

     /**
     * List all context keywords.
     *
     * @param  aoSess the ASAM ODS session.
     * @return a sequence of strings.
     * @throws CORBAFileServerException
     * with the following possible error codes:
     *    FILESERVER_CONNECTION_LOST
     *    FILESERVER_IMPLEMENTATION_PROBLEM
     *    FILESERVER_NOT_IMPLEMENTED
     *    FILESERVER_NO_MEMORY
     */
     DS_STRING listContext(
        in org::asam::ods::AoSession aoSess)
        raises (CORBAFileServerException);


     /**
     * Get the version of the CorbaFileServerIF.
     * Returns getVersion of CorbaFileServer.
     *
     * @throws CORBAFileServerException
     * with the following possible error codes:
     *    FILESERVER_CONNECTION_LOST
     *    FILESERVER_IMPLEMENTATION_PROBLEM
     *    FILESERVER_NOT_IMPLEMENTED
     *    FILESERVER_NO_MEMORY
     *
     * @return  The interface version of the CorbaFileServerIF.
     *
     */
     string getInterfaceVersion()
        raises (CORBAFileServerException);

     /**
     * Returns an array of long values representing the length
     * of the files that were provided in the String array.
     * The order of the long values must match with the order
     * of the filenames.
     *
     * @param aoSess the aoSession of the caller
     * @param names the String array of filenames for
     *              which to get the sizes
     * @return an Array of long values containing the file sizes
     * @throws CORBAFileServerException
     * with the following possible error codes:
     *    FILESERVER_CONNECTION_LOST
     *    FILESERVER_IMPLEMENTATION_PROBLEM
     *    FILESERVER_NOT_IMPLEMENTED
     *    FILESERVER_NO_MEMORY
     *
     *
     */
     LONG_ARRAY getSizes (
        in org::asam::ods::AoSession aoSess,
        in DS_STRING names)
        raises (CORBAFileServerException);

  };
```

```
}; // module generated
}; // module corbafileserver
}; // module highqsoft
}; // module com
```

Chapter 5

Plugin Support

The CORBAFileServer supports a plugin mechanism to run a data transfer plugin as a subsystem of the CFS itself. The plugin can transfer data in its own way, detached from the CFS. There is a listener system that allows the plugin to provide interested objects with events when file actions have been concluded.

5.1 Interface

The class that a plugin needs to implements is 'com.highqsoft.corbafileserver.plugin.TransportPlugin'. This class is instantiated with a default empty contructor when the CFS starts up.

the interface supplies 3 methods:

void init(Properties properties);

This method is used to initialize the plugin. After the method has returned, the plugin must be up and running to accept file operations.

void addPluginListener(PluginListener listener);

void removePluginListener(PluginListener listener);

These 2 methods can be used to register listeners for file action events on the server side. The CORBAFileServer does NOT register itself as a listener since the plugin is designed to work loosely coupled with the CFS.

5.2 Initialization of a plugin

To provide the CFS with the Plugin, there is a new Property 'Transport.Plugin.Class' which contains the class name of the plugin.

5.3 Default Implementation

There is a default implementation that can be used to transfer data. To use this plugin, set the a property '-Transport.Plugin.Class com.highqsoft.corbafileserver.plugin.impl.DefaultTransportPlugin' at the startup of the CFS. This plugin communicates with a basic TCP based protocol. The plugin supports PUT GET and DEL commands to write read and delete files at the server. It also uses the CFS's FilenameHandler and SecurityHandler to check for correct file assignment and security just like the normal CORBA methods. The plugin uses the same mechanics of getting a (Server)Socket as the CORBAFileServer. This means, the section about SSL Sockets in this manual applies to the Plugin as well. That way, secure file transfer can be done with the plugin. **IMPORTANT** For the SSL based transfer, the socket roles of server and client are reversed. That means, the client has to have the truststore settings and the CFS has to have the keystore setting!

5.3.1 Properties

The default implementation can be configured to control the behaviour. This is a list of supported properties:

Transport.Default.ServerPort

Must be an integer value. Provides the port where to Plugin should initiate a port to listen to requests from the clients. Defaults to 6689.

Transport.Default.AllowedCipherSuites

Must be a string value. Optional property that defines the cipher suites that are allowed by the server socket in the CFS to be used. The string can be a comma separated list of ciphers. Example:

```
-Transport.Default.AllowedCipherSuites TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_DSS_WITH_AES_128_CBC_SHA,
TLS_RSA_WITH_AES_128_CBC_SHA,TLS_KRB5_WITH_3DES_EDE_CBC_MD5,TLS_KRB5_WITH_3DES_EDE_CBC_SHA
```

Transport.Default.WorkersMax

The default implementation works with a pool of workers which handle the different Commands sent by clients. This property defines how many workers may work at any given moment. If more requests are received, they are stored in a FIFO Queue and handled as soon as other workers are done. The value must be an integer. The default value is 10.

5.4 Default Client

There is a client that comes with the implementation. This client can be used as a commend line version or

5.4.1 Usage of the client

the client can be started with a command line call:

Example:

```
$ java -jar <client.jar> COMMAND OPTIONS
```

The available COMMANDs are: PUT - Writes a file GET - Reads a file DEL - Deletes a file

For more information on how to use the client, start the client without arguments. This will print a help text with mroe details.

Example calls:

```
$ java -jar <client.jar> GET -file file://SERVER/dataroot/somefile.dat -localfile /home/mydir/file.dat -host somehost
```

This call fetches the file that was stored under the identifier `file://SERVER/dataroot/somefile.dat` and stores it locally at */home/mydir/file.dat*.

```
$ java -jar <client.jar> PUT -file /dataroot/somefile.dat -host somehost -port 7777
```

This call stores the local file */dataroot/somefile.dat* and connects to the plugin at host *somehost* and port *7777*

```
$ java -jar -Djavax.net.ssl.trustStore=cfs.jks -Djavax.net.ssl.trustStorePassword=cfs -jar
PluginClient.jar PUT -file file.dat -host 192.168.101.198 -CORBAFileServer.useSSL true
```

An example that uses SSL to transfer the file. The truststore is the file *cfs.jks*, the JKS pass is *cfs*. The transferred file is called *file.dat*. The host is the IP address 192.168.101.198. The setting CORBAFileServer.useSSL must be set since the client uses the same mechanics as the CFS for SSL.

5.4.2 Client API

The client can also be accessed programmatically. To do so, create an object of one of the following types:

- com.highqsoft.corbafileserver.plugin.impl.client.PUTClientCommand
- com.highqsoft.corbafileserver.plugin.impl.client.GETClientCommand
- com.highqsoft.corbafileserver.plugin.impl.client.DELClientCommand

You must provide the required information for a command in the *init(Properties p)* method. The javadoc of the commands contains the data that is necessary for the commands to work.

Chapter 6

Licensing

The CORBAFileServer is one of the products in the range of HighQSoft products that requires a license to run. There are the following properties defined for licensing configuration:

CORBAFileServer.LicenseLocation

This variable can either contain a directory (no URL, just the path) to a local directory where a license is provided. This case is used when the license is node-locked for the specific machine the CORBAFileServer is running on. The other option is to supply a machine in the syntax: **host@port**. This will tell the licensing mechanism of a License Manager where the licenses are stored. This usually occurs with floating licenses. The default value points to the local directory at

`'.../CORBAFilerServer-x.y.z.jar/../etc`

CORBAFileServer.LicenseListener

This variable can be used to introduce a separate listener for events on the license. The listener must implement the interface *com.highqsoft.corbafileserver.LicenseListener* and has several methods which provide information when a license was received or when it is about to expire. That information can be used to set up a management system to prevent the CFS from going down unexpectedly.

Chapter 7

Properties

The following table lists the usable properties for the plugin configuration.

CORBAFileServer.BufferSize

Datatype: **int**
DefaultValue: **100000**

The buffer size in bytes for file transfer.

CORBAFileServer.DebugLevel

Datatype: **int**
DefaultValue: **none**

To get verbose output set this value. Currently there is no convention at what is printed at what level.

CORBAFileServer.ExcludePattern

Datatype: **String**
DefaultValue: **none**

This property is can be used by the security handler. It is used by the default security handler ∗`com.highqsoft.corbafileserver.SecurityHandler` and checks the name coming from client against this pattern. If the pattern does matches, a security exception will be thrown.

CORBAFileServer.ExcludePatternFlags

Datatype: **String**
DefaultValue: **none**

This property is meaningful if the property*CORBAFileServer.ExcludePattern* is also available. It is a list of flags defined by the`java.util.regex.Pattern` and used by the method`compile(String, int)` . The flags must not be defined by the integer values. It can be defined directly by the field names optionally separated by vertical bar '|'. Example:`CASE_-INSENSITIVE | MULTILINE`

CORBAFileServer.FilenameHandler

Datatype: **classname**

DefaultValue: **com.highqsoft.corbafileserver.FilenameHandler**

The default filename handler. This class must fulfill the FilenameHandlerIF interface defined in this package.

CORBAFileServer.IncludePattern

Datatype: **String**
DefaultValue: **none**

This property is can be used by the security handler. It is used by the default security handler ∗`com.highqsoft.corbafileserver.SecurityHandler` and checks the name coming from client against this pattern. If the pattern doesn't matches, a security exception will be thrown.

CORBAFileServer.IncludePatternFlags

Datatype: **String**
DefaultValue: **none**

This property is meaningful if the property *CORBAFileServer.IncludePattern* is also available. It is a list of flags defined by the`java.util.regex.Pattern` and used by the method`compile(String, int)` . The flags must not be defined by the integer values. It can be defined directly by the field names optionally separated by vertical bar '|'. Example:`CASE_-INSENSITIVE | MULTILINE`

CORBAFileServer.LicenseLocation

Datatype: **String**
DefaultValue: ***path-to-CFS-jar/../etc***

The location of the license for the CORBAFileServer. The supported format of the string is either an absolute path to a directory with the license file(s) or a 'host' annotation of a machine where the license manager is running.

CORBAFileServer.NameServiceContext

Datatype: **string**
DefaultValue: **"com/highqsoft/"**

This property set the name service context. The name service context is the context in which servicename will be published. The context must ends with '/' character.

CORBAFileServer.NameServiceHost

Datatype: **String**
DefaultValue: **localhost**

The host name of the requested CORBA Name Service. This value has more priority than NameServiceHost, but is ignored if ORBInitRef is set. *This value is an application property.*

CORBAFileServer.NameServiceName

Datatype: **String**
DefaultValue: **NameService**

The name of the requested CORBA Name Service. This value has more priority than NameServiceName, but is ignored if ORBInitRef is set. *This value is an application property.*

CORBAFileServer.NameServicePort

Datatype: **String**
DefaultValue: **2809**

The port number of the requested CORBA Name Service. This value has more priority than NameServicePort, but is ignored if ORBInitRef is set. *This value is an application property.*

CORBAFileServer.PrintNameServiceIOR

Datatype: **boolean**
DefaultValue: **false**

This flags tells the server to print the NameService IOR to the console. This IOR can be used by the client to find the name service used by service.

CORBAFileServer.PublishInRoot

Datatype: **boolean**
DefaultValue: **true**

This flags tells the server to publish the CORBAFileServerIF in the root of the name service. When the CORBAFileServerIF is published in the root, the object will always have the kind ASAM-ODS.

CORBAFileServer.RegistrationFormat

Datatype: **String**
DefaultValue: **"{0}.{1}@{2}"**

This is the format string for the name that is used to register this service. There are three optional parameter available: 0 - The value of the property ServiceName 1 - The word*FileServer* 2 - The uppercase local hostname The default format creates a name like CORBAFileServer/FileServer

CORBAFileServer.RootDir

Datatype: **Directory**
DefaultValue: **$HOME/CORBAFileServer**

The root directory for the filehandler.

CORBAFileServer.SecurityActive

Datatype: **boolean**
DefaultValue: **true**

The boolean decides whether to call the security handler or not. if*true* it will be called.

CORBAFileServer.SecurityHandler

Datatype: **classname**
DefaultValue: **com.highqsoft.corbafileserver.SecurityHandler**

The default security handler. This class must fulfill the SecurityHandlerIF interface defined in this package.

CORBAFileServer.ServiceName

Datatype: **String**
DefaultValue: **CORBAFileServer**

The service to be used by the service. The name of for registration will be build by using the property CORBAFileServer.RegistrationFormat. *This value is an application property.*

CORBAFileServer.SocketTimeout

Datatype: **boolean**
DefaultValue: **2000**

The socket timeout in milliseconds.

CORBAFileServer.StdErr

Datatype: **File**
DefaultValue: **none**

The name of a file that service uses as standard error stream. This value can not set via a property files (parameter Properties), because it will be used directly after startup and before the property file is evaluated.

CORBAFileServer.StdOut

Datatype: **File**
DefaultValue: **none**

The name of a file that service uses as standard output stream. This value can not set via a property files (parameter Properties), because it will be used directly after startup and before the property file is evaluated.

CORBAFileServer.Terminator

Datatype: **classname**
DefaultValue: ***null***

The terminator that can be called by the client to terminate the transfer action.

CORBAFileServer.Terminator.Command

Datatype: **String**
DefaultValue: ***null***

The command to be executed during terminator process. The class uses the java.text.MessageFormat class to format the command line string. {0} the name of the file. {1} the application element id. {2} the instance element id. {3} the parameter coming from client. An example for a simple ant call may be: `ant -DCORBAFileServer.Filename="{0}" -F mybuild.xml`

CORBAFileServer.Terminator.WaitFor

Datatype: **boolean**
DefaultValue: **true**

Use this boolean to instruct to wait for the newly created process.

CORBAFileServer.URLPrefix

Datatype: **String**
DefaultValue: file://"+**Helper.getHostname().toUpperCase()**

This is the prefix that the file handler uses to create the URL of the file.

CORBAFileServer.WasteDir

Datatype: **Directory**
DefaultValue: **$HOME/CORBAFileServer/waste**

The root directory for the filehandler where to move the waste.

CORBAFileServer.useSSL

Datatype: **boolean**
DefaultValue: **false**

Tells the CORBAFileServer to initialize SSLSockets instead of normal Sockets. The Java SSL stack must be set up correctly by the IT administrator.

CORBAProtocol

Datatype: **String**
DefaultValue: **""**

The CORBA protocol in the corbaloc (CORBA locator) to find the name service. The default of CORBA is**iiop** , which is identical with an empty string "". For IIOP over SSL the protocol must be**ssliop** . This value is ignored if ORBInitRef is set. *This value is an application property.*

NameServiceHost

Datatype: **String**
DefaultValue: **localhost**

The host name of the requested CORBA Name Service. This value has less priority than CORBAFileServer.NameServiceHost, both are ignored if ORBInitRef is set. *This value is an application property.*

NameServiceName

Datatype: **String**
DefaultValue: **NameService**

The name of the requested CORBA Name Service. This value has less priority than CORBAFileServer.NameServiceName, both are ignored if ORBInitRef is set. *This value is an application property.*

NameServicePort

Datatype: **String**
DefaultValue: **2809**

The port number of the requested CORBA Name Service. This value has less priority than CORBAFileServer.NameServicePort, both are ignored if ORBInitRef is set. *This value is an application property.*

ODSFilenameHandler.FileRoot

Datatype: **String**
DefaultValue: **FILE_ROOT_EXTREF**

This is the symbol of the ODS Server where the files will be stored.

ODSFilenameHandler.URLPrefix

Datatype: **String**
DefaultValue: `file://`

This is the prefix that the file handler uses to create the URL of the file.

ODSFilenameHandler.UseSessionCache

Datatype: **boolean**
DefaultValue: **true**

Use a cache for the ODSFilenameURL, the cache will reduce the requests from Filename-Handler to the ODS Server. Without a cache for each file, handled in the same session, the symbols are loaded from the ODS server. With the cache the symbols are loaded only once. When the symbols changes during the session , you can not use the cache. ∗**This property is obsolete, because caching leads to synchronization issues.**

ORBInitRef

Datatype: **String**
DefaultValue: **none**

The full CORBA locator. *If this value is set, the values of (CORBAFile-Server.)NameServiceName, NameServicePort, NameServiceHost and CORBAProtocol are ignored. This value is an application property.* Example: Nameser-vice=corbaloc::1.2:MyServicePort/NameService

Properties

Datatype: **filename**
DefaultValue: ***none***

A file that contains the properties. The properties defined directly as parameters have higher priority.

Chapter 8

Examples

A JUnit test program shows how connect to the server and how to control the different use-cases.

```
package test.highqsoft.corbafileserver;


import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.HashMap;

import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

import org.asam.ods.AoSession;

import com.highqsoft.corbafileserver.Helper;
import com.highqsoft.corbafileserver.SocketReader;
import com.highqsoft.corbafileserver.SocketWriter;
import com.highqsoft.corbafileserver.StreamFactory;
import com.highqsoft.corbafileserver.generated.CORBAFileServerException;
import com.highqsoft.corbafileserver.generated.CORBAFileServerIF;

/**
 * Test class to test the access to the ORBAFileServer.
 *
 * @author Andreas Hofmann
 * @version $Revision: 1.6 $
 * @since $Date: 2015/05/20 13:06:31 $
 */

/*
 * Modification History:
 *
 * $Log: TestBase.java,v $
 * Revision 1.6  2015/05/20 13:06:31  karst
 * Update Copyrights to current.
 *
 * Revision 1.5  2013/06/04 07:38:07  karst
 * Order the testcases.
 *
 * Revision 1.4  2009/01/05 10:04:54  karst
 * Modify copyright to 2009
 *
 * Revision 1.3  2007/04/23 06:58:46  andy
 * Use the socket connection for this test.
 *
 * Revision 1.2  2006/10/06 06:42:40  andy
 * Add the logger as parameter to several methods.
 *
 * Revision 1.1  2006/01/20 15:55:19  andy
 * Initial CVS-revision.
 *
 */
public class TestBase extends TestCase {

   protected CORBAFileServerIF server=null;
   protected AoSession aoSession=null;

   // Map with the threads.
   static HashMap threadRegistry = new HashMap();

   public TestBase(String method) {
      super(method);
   }

   public TestBase() {
      super();
   }



   public void setUp() {

      StreamFactory f = StreamFactory.getInstance();
      f.setProperties(System.getProperties());
//      server = f.getService(f.getServiceName(Helper.getHostname()));
//      server = f.getService(f.getServiceName("AUDIINSA0004A"));
      try {
         String list[] = f.listServices(".*@"+Helper.getHostname(), null);//System.getProperty("CORBAFileServer.NameServ

         assertNotNull(list);
         assertEquals(list.length, 2);

         server = f.getService(list[0], null);
         System.out.println("Server is available."+server);

         aoSession = null;
      } catch (CORBAFileServerException fse) {
         throw new AssertionError (Helper.exceptionToString(fse));
      }
   }

   public void tearDown() {

   }

   public void testContext() {
      // List all context variables.
      try {
         String arr[] = server.listContext(aoSession);
         for (int i=0; i<arr.length; i++) {
            System.out.println (arr[i]+": "+server.getContext(aoSession, arr[i]));
         }
      } catch (CORBAFileServerException fse) {
         throw new AssertionError (Helper.exceptionToString(fse));
      }

      try {
         String val = "100000";
         server.setContext(aoSession, "CORBAFileServer.BufferSize", val);
         StreamFactory.getInstance().setProperty("CORBAFileServer.BufferSize", val);
         assertEquals(server.getContext(aoSession, "CORBAFileServer.BufferSize"), val);
         System.out.println ("The value of the context CORBAFileServer.BufferSize: "+
               server.getContext(aoSession, "CORBAFileServer.BufferSize"));

      } catch (CORBAFileServerException fse) {
         throw new AssertionError (Helper.exceptionToString(fse));
      }
   }

   public void testFileTransfer() {
      long time;
      File file;
      File clientFile;
      String serverURL;

      // Create the temporary data.
      try {
         file = File.createTempFile("CFSTestBase",".dat");
         file.deleteOnExit();
         FileOutputStream fos = new FileOutputStream(file);
         byte[] arr = new byte[1024*1024];
         for (int i=0; i<20; i++) {
            fos.write(arr);
         }
         fos.close();
         System.out.println("Testdata in file <"+file.toString()+"> created with size "+file.length()+".");

         System.out.println ("Server buffer size for FileTransfer test: "+server.getContext(aoSession, "CORBAFileServer

      } catch (CORBAFileServerException fse) {
         throw new AssertionError (Helper.exceptionToString(fse));
      } catch (IOException ioe) {
         throw new AssertionError (ioe);
      }
/*
      // Save file to server via CORBA.
      time = System.currentTimeMillis();
      try {
         InputStream is = new InputStream(file);
         InputStreamIF co = Helper.enableInputStream(is);
         serverURL = server.save(aoSession, "MassData.dat", "", co);
         System.out.println("Massdata are saved to server: <"+serverURL+">");

      } catch (FileNotFoundException fnf) {
         throw new AssertionError (fnf);
      } catch (CORBAFileServerException fse) {
         System.err.println(fse.getLocalizedMessage()+ " " +fse.reason);
         throw new AssertionError (Helper.exceptionToString(fse));
      }
      System.out.println ("elapsed time for writing via CORBA: "+(System.currentTimeMillis()-time)+" msec");
*/
      // Save file to server via Socket.
      time = System.currentTimeMillis();
      try {
         SocketReader reader = new SocketReader(file, SocketReader.DEFAULT_PORT, Helper.DEFAULT_SOCKET_TIMEOUT);
         reader.start();
         serverURL = server.saveBySocket(aoSession, file.getName(), "", Helper.getHostname(), SocketReader.DEFAULT_PORT
         reader.join();
         System.out.println("Massdata are saved to server: <"+serverURL+">");
      } catch (FileNotFoundException fnf) {
         throw new AssertionError (fnf);
      } catch (CORBAFileServerException fse) {
         System.err.println(fse.getLocalizedMessage()+ " " +fse.reason);
         throw new AssertionError (Helper.exceptionToString(fse));
      } catch (Throwable t) {
         throw new AssertionError (t.getClass().getName()+" "+t.getMessage());
      }
      System.out.println ("elapsed time for writing via socket: "+(System.currentTimeMillis()-time)+" msec");

/*
      // Read the file from server CORBA.
      time = System.currentTimeMillis();
      try {

         InputStreamIF co = server.read(aoSession, serverURL);
         clientFile = File.createTempFile("ClientFile",".dat");
         clientFile.deleteOnExit();
         Helper.readFromStream(co, clientFile);
         co.close();
         System.out.println("Massdata are read from server: <"+clientFile+">");

      } catch (CORBAFileServerException fse) {
         throw new AssertionError (Helper.exceptionToString(fse));
      } catch (IOException ioe) {
         throw new AssertionError (ioe);
      }
      System.out.println ("elapsed time for reading via CORBA: "+(System.currentTimeMillis()-time)+" msec");
*/

      // Read the file from server via socket.
      time = System.currentTimeMillis();
      try {
         clientFile = File.createTempFile("ClientFile",".dat");
         clientFile.deleteOnExit();
         SocketWriter writer = new SocketWriter(clientFile, SocketWriter.DEFAULT_PORT, Helper.DEFAULT_SOCKET_TIMEOUT);
         writer.start();
         server.getBySocket(aoSession, serverURL, Helper.getHostname(), SocketWriter.DEFAULT_PORT);
         writer.join();
         System.out.println("Massdata are read from server: <"+clientFile+">");

      } catch (CORBAFileServerException fse) {
         throw new AssertionError (Helper.exceptionToString(fse));
      } catch (IOException ioe) {
         throw new AssertionError (ioe);
      } catch (Throwable t) {
         throw new AssertionError (t.getClass().getName()+" "+t.getMessage());
      }
      System.out.println ("elapsed time for reading via Socket: "+(System.currentTimeMillis()-time)+" msec");

      // Delete the file on server side.
      time = System.currentTimeMillis();
      try {
         server.delete(aoSession, serverURL);
         System.out.println("Massdata are deleted on server: <"+serverURL+">");
         clientFile.delete();
         System.out.println("Massdata are deleted on client: <"+clientFile+">");
         file.delete();
         System.out.println("Testdata are deleted: <"+file+">");
      } catch (CORBAFileServerException fse) {
         throw new AssertionError (Helper.exceptionToString(fse));
      }
      System.out.println ("elapsed time for deleting: "+(System.currentTimeMillis()-time)+" msec");
   }

   public static Test suite() {
       return new TestSuite(TestBase.class);
   }

   public static void main(String[] args) {
       junit.textui.TestRunner.run(suite());
   }
}
```

Chapter 9

Modification History

Date	**Description**	**Author**
17 January 2005	Grundversion	Andreas Hofmann
24 May 2007	Translated from german to english.	Andreas Hofmann

Index

com.highqsoft.corbafileserver.TerminateProcess, 7

delete, 3

example, 37

FilenameHandler, 5

Interface, 11
Introduction, 1

LicenseListener, 29
Licensing, 29

MS-Windows System Service, 9

ODSFilenameHandler, 6

Plugin, 25
properties, 31

read, 3
rename, 4

save, 3
Secure Transfer, 9
SecurityHandler, 6
SSL, 9
starting from command line, 9

Terminator, 7
TransportPlugin, 25