SDK Loader User Manual

S. Magini, P. Pesciullesi, L. Potiti, G. Virdis

{maginis, pesciul, potiti, virdis}@di.unipi.it

Index

1 Overview

The Software Development Kit (SDK) is intended to enable software developers to create applications that interacts with the Loader ASSIST. The Loader ASSIST is an application that serves for launching applications ASSIST or component ASSIST, CCM or Web Service.
For further information and one detailed description, refer to DescrizioneLoader.pdf.
The SDK consists of a C++ library and reusable programming Java objects, which, by means of a published Application Programming Interface (API), enable the application software to interact with the Loader ASSIST.

2 Installation

The SDK is included in the version 1.3 of ASSIST. At the moment for a correct installation it is necessary to compile ASSIST, refer to CompileAssist.pdf in the doc directory. The environment variable ASTCC_ROOT points out the directory of installation of ASSIST.

2.1 Library C++ (libclient.a)

Requirement:
libxml2 2.6.16 or following version.

For the generation of the library C++ libClient.a is enough to position him in the directory:
```
> cd $(ASTCC_ROOT)/Loader/ClientLoader
```
and digitare the command
```
> make
```

To use the library is necessary to link it in phase of compilation and to include the necessary headers in the source code. The files .h are found in the directory $(ASTCC_ROOT)/Loader/ClientLoader.

2.2 JAVA Classes (ClientAdp, TcpClient)

Requirement:

jdom.jar	($(ASTCC_ROOT)/Loader/lib)
xercesImpl.jar	($(ASTCC_ROOT)/Loader/lib)
xmlParserAPIs.jar	($(ASTCC_ROOT)/Loader/lib)
Loader.jar	($(ASTCC_ROOT)/Loader/build/lib)

For the correct operation it is necessary to insert in the CLASSPATH the necessary files .jar.

2.3 JAVA Classes (ClientAdp, WSClient)

activation.jar	($(ASTCC_ROOT)/Loader/lib)
axis.jar	($(ASTCC_ROOT)/Loader/lib)
jaxrpc.jar	($(ASTCC_ROOT)/Loader/lib)
mail.jar	($(ASTCC_ROOT)/Loader/lib)

For the correct operation it is necessary to insert in the CLASSPATH the necessary files .jar.

3 SDK Usage

This chapter provides detailed information regarding the usage of the SDK. It is intended to be read sequentially, and should be done so before attempting to utilize the SDK. The general design of the application is introduced and detailed examples are given.
For detailed API reference, syntax, arguments, and error codes, please refer SDK APIs.

3.1 Assist Application

To require the services offered by the Loader is necessary to install a client. The class ClientAdp defines the methods of the Loader that a client can use. Currently two implementations exist in JAVA of the class ClientAdp (the class TcpClient and the class WSClient) and an implementation in C++ (the class TcpClient).
The first example is a complete ASSIST application that loads application, it performs the application and it attends the results.

```
import it.unipi.di.client.*;

public class ApplicazioneAssist {

    public ApplicazioneAssist() {};

    public static void main( String[] args ) {

       String server  = "orione.di.unipi.it";
       int    port    = 12500;
       String applXml = "/tmp/ast.out.xml";

       ClientAdp    cl        = new TcpClient (server, port);
       StringBuffer result    = new StringBuffer();
       StringBuffer libHandle = new StringBuffer();
       StringBuffer runHandle = new StringBuffer();
       StringBuffer stringaXML= new StringBuffer();
       boolean err = true;

       // Loads the description of the machines managed by the Loader
       err = cl.CaricaCluster(result);
       if (!err) System.exit(-1);

       // Returns the information on the machines managed by the Loader
       err = cl.GetInfo(stringaXML);
       if (!err) System.exit(-1);

       // Loads the ASSIST program description and returns the associated identifier
       err = cl.LoadAA(applXml, result, libHandle);
       if (!err) System.exit(-1);

       // Runs an ASSIST program associated to a libHandle and returns the associated identifier
       int libHandleInt = Integer.parseInt(libHandle.toString());
       err = cl.ExecAA( libHandleInt, result, runHandle);
       if (!err) System.exit(-1);

       // Return only when ASSIST program is terminated and creates a file that includes the
execution
       int runHandleInt = Integer.parseInt(runHandle.toString());
       String pack = "/tmp/risultati.aar";
       err = cl.GetResult(runHandleInt, pack, result);
       if (!err) System.exit(-1);

       // Deletes the information paired with libHandle from memory
       err = cl.Close(libHandleInt), result);
       if (!err) System.exit(-1);
    }
}
```

Figura 1

The first operation to be done is to install the client. In this case (Figure 1) we have chosen the implementation that communicates with the Loader through socket:

```
ClientAdp    cl        = new TcpClient (server, port);
```

server is the host name where the Loader is running and port is the door where the Loader is waiting.
Instead the first method is CaricaCluster because the Loader without a loaded architecture cannot perform some other command. Optionally we can access information on the architecture through the command GetInfo. With the command LoadAA we load the application in memory and we get the identifier used in the following ExecAA. In this moment the application is in execution. To recover the results of the execution is necessary to recall the command GetResult. Through the command Close we free the memory of the Loader from the loaded application. The implementation C++ is practically equal (Figure 2).

```
#include <TcpClient.h>
#include <stdlib.h>

int main(int argc, char *argv[]){


  string server  = "orione.di.unipi.it";
  int    port    = 12500;
  string applXml = "/tmp/ast.out.xml";

  string result;
  string libHandle;
  string runHandle;
  string stringaXML;
  bool err = true;

  ClientAdp* cl = new TcpClient(server, port);

  // Loads the description of the machines managed by the Loader
  err = cl->CaricaCluster(&result);
  if (!err) exit(-1);

  // Returns the information on the machines managed by the Loader
  err = cl->GetInfo(&stringaXML);
  if (!err) exit(-1);

  // Loads the ASSIST program description and returns the associated identifier
  err = cl->LoadAA(applXml, &result, &libHandle);
  if (!err) exit(-1);

  // Runs an ASSIST program associated to a libHandle and returns the associated identifier
  int libHandleInt = atoi(libHandle.c_str());
  err = cl->ExecAA( libHandleInt, &result, &runHandle);
  if (!err) exit(-1);

  // Return only when ASSIST program is terminated and creates a file that includes the execution
  int runHandleInt = atoi(runHandle.c_str());
  string pack = "/tmp/risultati.aar";
  err = cl->GetResult(runHandleInt, pack, &result);
  if (!err) exit(-1);

  // Deletes the information paired with libHandle from memory
  err = cl->Close(libHandleInt, &result);
  if (!err) exit(-1);

  delete cl;

  return 0;
}
```

Figura 2

3.2 Assist Component

The execution of a ASSIST component is very similar to the execution of an ASSIST application (Figure 3 and Figure 4). Difference is in the use of LoadAC and ExecAC. The method LoadAC requires as parameter a aar file that contains information on the component. This file aar is producted directly from the new CompComp instrument and contains the necessary files that describe the component, the binary files of ASSIST component and in case the binary files of the hoc processes, the libraries needed by the component and the configuration files for the hoc processes and the dynamics reconfiguration. Also the method ExecAC requires a aar file as parameter. This file is optional and memorizes the possible input data of the component. Don't use it is enough to pass an empty string as parameter.

```
import it.unipi.di.client.*;

public class ComponenteAssist {

    public ComponenteAssist() {};

    public static void main( String[] args ) {

        String server  = "orione.di.unipi.it";
        int    port    = 12500;
        String aar     = "/tmp/descrizione.aar";
        String datiaar = "/tmp/dati.aar";

        ClientAdp    cl        = new TcpClient (server, port);
        StringBuffer result    = new StringBuffer();
        StringBuffer libHandle = new StringBuffer();
        StringBuffer runHandle = new StringBuffer();
        StringBuffer stringaXML= new StringBuffer();
        boolean err = true;

        // Loads the description of the machines managed by the Loader
        err = cl.CaricaCluster(result);
        if (!err) System.exit(-1);

        // Returns the information on the machines managed by the Loader
        err = cl.GetInfo(stringaXML);
        if (!err) System.exit(-1);

        // Loads the ASSIST program description and returns the associated identifier
        err = cl.LoadAC(aar, result, libHandle);
        if (!err) System.exit(-1);

        // Runs an ASSIST program associated to a libHandle and returns the associated identifier
        int libHandleInt = Integer.parseInt(libHandle.toString());
        err = cl.ExecAC(libHandleInt, datiaar, result, runHandle, stringaXml);
        if (!err) System.exit(-1);

        // Return only when ASSIST program is terminated and creates a file that includes the
execution
        int runHandleInt = Integer.parseInt(runHandle.toString());
        String pack = "/tmp/risultati.aar";
        err = cl.GetResult(runHandleInt, pack, result);
        if (!err) System.exit(-1);

        // Deletes the information paired with libHandle from memory
        err = cl.Close(libHandleInt), result);
        if (!err) System.exit(-1);
    }
}
```
Figura 3

```
#include <TcpClient.h>
#include <stdlib.h>

int main(int argc, char *argv[]){


  string server  = "orione.di.unipi.it";
  int    port    = 12500;
  string aar     = "/tmp/descrizione.aar";
  string datiaar = "/tmp/dati.aar";

  string result;
  string libHandle;
  string runHandle;
  string stringaXML;
  bool err = true;

  ClientAdp* cl = new TcpClient(server, port);

  // Loads the description of the machines managed by the Loader
  err = cl->CaricaCluster(&result);
  if (!err) exit(-1);

  // Returns the information on the machines managed by the Loader
  err = cl->GetInfo(&stringaXML);
  if (!err) exit(-1);

  // Loads the ASSIST program description and returns the associated identifier
  err = cl->LoadAC(aar, &result, &libHandle);
  if (!err) exit(-1);

  // Runs an ASSIST program associated to a libHandle and returns the associated identifier
  int libHandleInt = atoi(libHandle.c_str());
  err = cl->ExecAC( libHandleInt, datiaar, &result, &runHandle, stringaXml);
  if (!err) exit(-1);

  // Return only when ASSIST program is terminated and creates a file that includes the execution
  int runHandleInt = atoi(runHandle.c_str());
  string pack = "/tmp/risultati.aar";
  err = cl->GetResult(runHandleInt, pack, &result);
  if (!err) exit(-1);

  // Deletes the information paired with libHandle from memory
  err = cl->Close(libHandleInt, &result);
  if (!err) exit(-1);

  delete cl;

  return 0;
}
```
Figura 4

3.3 *Assist Component with Repository*

The Assist component can be also memorized in the Loader. Every component is recorded an only time: in comparison to the preceding version the file aar is transferred therefore an only time. A recorded component can only be used by the clients that they possess the uuid released during the recording. In comparison to the preceding version it is necessary to use the command Registry, for the component's memorization in the Loader and the command LOADACR in substitution of the command LOADAC. If we want to use an already recorded component naturally not the Registry is necessary.

```
import it.unipi.di.client.*;

public class ComponenteAssistRepository {

    public ComponenteAssistRepository() {};

    public static void main( String[] args ) {

       String server  = "orione.di.unipi.it";
       int    port    = 12500;
       String aar     = "/tmp/descrizione.aar";
       String datiaar = "/tmp/dati.aar";

       ClientAdp    cl        = new TcpClient (server, port);
       StringBuffer result    = new StringBuffer();
       StringBuffer libHandle = new StringBuffer();
       StringBuffer runHandle = new StringBuffer();
       StringBuffer stringaXML= new StringBuffer();
       StringBuffer uuid      = new StringBuffer();
       boolean err = true;

       // Carica in memoria le informazioni sulle macchine gestite dal Loader
       err = cl.CaricaCluster(result);
       if (!err) System.exit(-1);

       // Restituisce le informazioni sull'architettura gestita dal Loader
       err = cl.GetInfo(stringaXML);
       if (!err) System.exit(-1);

       // Registra il componente nel repository del Loader e restituisce l'uuid
       err = cl.Registry(aar, result, uuid);
       if (!err) System.exit(-1);

       // Carica in memoria le informazioni sulla componente ASSIST relativa all'uuid
       // e restituisce l'handle associato
       err = cl.LoadACR(uuid.toString(), result, libHandle);
       if (!err) System.exit(-1);

       // Inizializza e lancia la componente ASSIST associata associata al libHandle
       int libHandleInt = Integer.parseInt(libHandle.toString());
       err = cl.ExecAC(libHandleInt, datiaar, result, runHandle, stringaXml);
       if (!err) System.exit(-1);

       // Restituisce i risultati dell'esecuzione associata al runHandle
       int runHandleInt = Integer.parseInt(runHandle.toString());
       String pack = "/tmp/risultati.aar";
       err = cl.GetResult(runHandleInt, pack, result);
       if (!err) System.exit(-1);

       // Scarica dalla memoria tutte le informazioni associate al libHandle
       err = cl.Close(libHandleInt), result);
       if (!err) System.exit(-1);
    }
}
```

Figura 5

```
#include <TcpClient.h>
#include <stdlib.h>

int main(int argc, char *argv[]){


  string server  = "orione.di.unipi.it";
  int    port    = 12500;
  string aar     = "/tmp/descrizione.aar";
  string datiaar = "/tmp/dati.aar";


  string result;
  string libHandle;
  string runHandle;
  string stringaXML;
  string uuid;
  bool err = true;

  ClientAdp* cl = new TcpClient(server, port);

  // Carica in memoria le informazioni sulle macchine gestite dal Loader
  err = cl->CaricaCluster(&result);
  if (!err) exit(-1);

  // Restituisce le informazioni sull'architettura gestita dal Loader
  err = cl->GetInfo(&stringaXML);
  if (!err) exit(-1);

  // Registra il componente nel repository del Loader e restituisce l'uuid
  err = cl->Registry(aar, &result, &uuid);
  if (!err) exit(-1);

  // Carica in memoria le informazioni sulla componente ASSIST relativa all'uuid
  // e restituisce l'handle associato
  err = cl->LoadACR(uuid, &result, &libHandle);
  if (!err) exit(-1);

  // Inizializza e lancia la componente ASSIST associata associata al libHandle
  int libHandleInt = atoi(libHandle.c_str());
  err = cl->ExecAC( libHandleInt, datiaar, &result, &runHandle, stringaXml);
  if (!err) exit(-1);

  // Restituisce i risultati dell'esecuzione associata al runHandle
  int runHandleInt = atoi(runHandle.c_str());
  string pack = "/tmp/risultati.aar";
  err = cl->GetResult(runHandleInt, pack, &result);
  if (!err) exit(-1);

  // Scarica dalla memoria tutte le informazioni associate al libHandle
  err = cl->Close(libHandleInt, &result);
  if (!err) exit(-1);

  delete cl;

  return 0;
}
```
Figura 6

The management of the recorded components happens through:
Unregistry: it eliminates the component from the Loader.
UnregistryR: it eliminates the component from the Loader and from the other known Loader.
GetComponent: it recovers the component if it is present in the Loader.
GetComponentR: it recovers the component if it is present in one of the known Loader.
GetDescrComponent: it recovers the description of the component if it is present in the Loader.
GetDescrComponentR: it recovers the description of the component if it is present in one of the known Loader
To its inside Every Loader has a list of other Loader that it can contact. This list represents the known Loaders.

```
String uuid      = "12346789";
err  = cl.Unregistry(uuid, result);
if (!err) System.exit(-1);

String uuid      = "12346789";
err  = cl.UnregistryR(uuid, result);
if (!err) System.exit(-1);

String uuid      = "12346789";
String nomefile  = "/tmp/nomefile.aar";
err  = cl.GetComponent(uuid, nomefile, result);
if (!err) System.exit(-1);

String uuid      = "12346789";
String nomefile  = "/tmp/nomefile.aar";
err  = cl.GetComponentR(uuid, nomefile, result);
if (!err) System.exit(-1);

String uuid      = "12346789";
String nomefile  = "/tmp/descrizione.xml";
err  = cl.GetDescrComponent(uuid, nomefile, result);
if (!err) System.exit(-1);

String uuid      = "12346789";
String nomefile  = "/tmp/descrizione.xml";
err  = cl.GetDescrComponentR(uuid, nomefile, result);
if (!err) System.exit(-1);
```

3.4 Web Service Interface

If the Web Service interface of the Loader is active it is possible to use the communication through Web Service between Client and Loader, please refer ChannelAdaptor_Webservice.pdf. It is enough to replace the class Tcpclient with the class WSClient in the implementation Java. To the moment an implementation C++ doesn't exist.

```
ClientAdp    cl        = new WSClient (server, port);
```

server is the host name where the Web Service is running and port is the door where the Web Service is waiting.

3.5 Application Manager Protocol

Some commands are used for implementing the protocol of communication among The Application Manager and the Loader. The Application Manager requires the throwing of some processes to the Loader. These processes send information to the Loader through SDK.

```
String portStrategy    = "12345";
command = "PORT_S:"+portStrategy;
err  = cl.GenericCommand(runHandle, command, result);
if (!err) System.exit(-1);


String portMaster      = "12346";
command = "PORT_M:"+portMaster;
err  = cl.GenericCommand(runHandle, command, result);
if (!err) System.exit(-1);


String HOCMaster_host      = "andromeda.di.unipi.it";
String HOCMaster_port      = "12347"
command = "HOCM:"+HOCMaster_host+":"+HOCMaster_port;
err  = cl.GenericCommand(runHandle, command, result);
if (!err) System.exit(-1);


String HOCSlave_host      = "capraia.di.unipi.it";
String HOCSlave_port      = "12348"
command = "HOCS:"+HOCSlave_host+":"+HOCSlave_port;
err  = cl.GenericCommand(runHandle, command, result);
if (!err) System.exit(-1);


String HOCStream_host      = "pegaso.di.unipi.it";
String HOCStream_port      = "12349"
command = "HOCSTR:"+HOCStream_host+":"+HOCStream_port;
err  = cl.GenericCommand(runHandle, command, result);
if (!err) System.exit(-1);
```

The Application Manager to increase the number of resources can invoke the command AddProcess

```
int nproc = 1;
String excutableName = "ND001__Integra_vpm";
err  = cl.AddProcess(nproc, runHandle, executableName, result);
if (!err) System.exit(-1);
```

The method KillProcess can be invoked by the Application Manager when the execution of a component is finished. In this case the trials hocstream is finished launched by the Loader for that component:

```
err = cl.KillProcess(runHandle, result);
if (!err) System.exit(-1);
```

4 SDK APIs

This chapter provides detailed information regarding the methods exposed by the APIs included in the SDK. It is not intended to be read sequentially, but rather be used as a reference. Each method has a dedicated description explaining its syntax, arguments, results, and possible error conditions. For detailed usage information and examples, please refer to SDK Usage.

4.1 Carica Cluster

Description
Loads the description of the machines managed by the Loader when the Loader uses the Ssh mechanism

Syntax
```
bool CaricaCluster  (string* result)
boolean CaricaCluster  (StringBuffer result);
```

Parameters
result
[out] Receives the result of the command

Return Values
If the function succeds, the return value is true else false

Remarks
The CaricaCluster methods should be the first method invoked when using the SDK
Caricacluster must not have called if the Loader uses mechanisms owners of Globus.In the case the execution is called it doesn't have effect.

4.2 GetInfo

Description
Returns the information on the machines managed by the Loader.

Syntax
```
bool GetInfo        (string* stringaXML)
boolean GetInfo    (StringBuffer stringaXML);
```

Parameters
stringaXML
[out] Receives the information of the architecture managed by the Loader

Return Values
If the function succeds, the return value is true else false

Remarks
If the Loader is installed on the front-end of a private net it is possible that received information are partial, it is possible that the administrator hides addresses IP to the outside.
GetInfo returns an empty string if the Loader uses mechanisms owners of Globus

4.3 LoadAA

Description
Loads the ASSIST program description and returns the associated identifier

Syntax
```
bool LoadAA         (string applXml, string* result, string* libHandle)
boolean LoadAA     (String applXml, StringBuffer result, StringBuffer libHandle);
```

Parameters
applXml
[in] points out the file that it describes the ASSIST application
result
[out] Receives the result of the command
libHandle
[out] receives the associated identifier

Return Values
If the function succeds, the return value is true else false

Remarks
the file that it describes the ASSIST application is automatically produced by the compiler astCC.

4.4 ExecAA

Description
Runs an ASSIST program associated to a libHandle and returns the associated identifier

Syntax
```
bool ExecAA         (int libHandle,  string* result, string* runHandle)
boolean ExecAA     (int libHandle,  StringBuffer result, StringBuffer runHandle);
```

Parameters
libHandle
[in] identifies the application to run
result
[out] Receives the result of the command
runHandle
[out] receives the associated identifier

Return Values
If the function succeds, the return value is true else false

Remarks
this method doesn't expect termination of the application but immediately returns

4.5 LoadAC

Description

Loads the aar file containing the ASSIST component and returns the associated identifier

Syntax

```
bool LoadAC         (string aar,     string* result, string* libHandle)
boolean LoadAC     (String aar,     StringBuffer result, StringBuffer libHandle);
```

Parameters

aar
[in] points out the file that it describes the ASSIST component
result
[out] Receives the result of the command
libHandle
[out] receives the associated identifier

Return Values

If the function succeds, the return value is true else false

Remarks

The method LOADAC performs the same operations of the method LOADAA: the file xml that describes the application is found inside the file aar

4.6 ExecAC

Description

Runs a ASSIST component associated to a libHandle and returns the associated identifier

Syntax

```
bool ExecAC         (int libHandle,  string  aar,    string* result, string* runHandle,
string* stringXml)
boolean  ExecAC       (int  libHandle,   String          aar,      StringBuffer  result,
StringBuffer runHandle, StringBuffer stringXml);
```

Parameters

libHandle
[in] identifies the component to run
aar
[in] contains the input files of component
result
[out] Receives the result of the command
runHandle
[out] receives the associated identifier

Return Values

If the function succeds, the return value is true else false

Remarks

The parameter aar has to be empty string if the component ASSIST doesn't require entry file.

4.7 GetResult

Description

Return only when ASSIST program is terminated and creates a file that includes the execution results

Syntax

```
bool GetResult      (int runHandle,  string nomefile_ass, string* result);
boolean GetResult  (int runHandle,  String nomefile_ass, StringBuffer result);
```

Parameters

runHandle
[in] identifies the execution
nomefile_ass
[in] it points out the file where to memorize the results
result
[out] Receives the result of the command

Return Values

If the function success, the return value is true else false

4.8 KillProcess

Description

kills all the processes launched during the execution associated to a runHandle

Syntax

```
bool KillProcess    (int runHandle,  string* result)
boolean KillProcess     (int runHandle,  StringBuffer result);
```

Parameters

runHandle
[in] identifies the execution
result
[out] Receives the result of the command

Return Values

If the function success, the return value is true else false

4.9 Close

Description
Deletes the information paired with libHandle from memory.

Syntax
```
bool Close        (int libHandle, string* result)
boolean Close      (int libHandle,  StringBuffer result);
```

Parameters
libHandle
[in] identifies the application/component to delete
result
[out] Receives the result of the command

Return Values
If the function success, the return value is true else false

Remarks
It removes also the execution informations associated to runHandle paired to libHandle. Furthermore deletes the directory and the temporary file created during execution.

4.10 AddProcess

Description
Allows to add resources to the application ASSIST

Syntax
```
bool AddProcess       (int numProcess, int runHandle,   string nomeEseguibile, string* result)
boolean AddProcess (int numProcess, int runHandle, String nomeEseguibile, StringBuffer result);
```

Parameters
numProcess
[in] number of resources to be added
runHandle
[in] identifies the execution
nomeEseguibile
[in] identifies the resource
result
[out] Receives the result of the command

Return Values
If the function success, the return value is true else false

Remarks
this service can be invoked only by the strategy process of the application: it is the Application Manager through his analyses that checks the contract of performance and it decides to increase the number of resources. Currently the strategy is not able to automatically require the increase of the resources, for this motive the service is also available for generic client.

4.11 GenericCommand

Description
Sends information to the Loader.

Syntax
```
bool GenericCommand (int runHandle,  string generic_command, string* result);
boolean GenericCommand  (int runHandle,  String generic_command, StringBuffer result);
```

Parameters
runHandle
[in] identifies the execution
generic_command
[in] contains the information that the process wants to communicate
result
[out] Receives the result of the command

Return Values
If the function success, the return value is true else false

Remarks
Currently the Loader can receive following information (generic_command):
PORT_S:port
where port is the door opened by the strategy process of the application ASSIST
PORT_M:port
where port is the door opened by the master process of the application ASSIST
HOCM:host:port
where host and port are respectively the host on which the hoc master process has been launched and the door opened by the hoc master process to interact with the other hoc processes
HOCS:host:port
where host and port are respectively the host on which a hoc process has been launched and the door opened by the hoc process to interact with the ASSIST processes
HOCSTR:host:port
where host and port are respectively the host on which a hocStream process has been launched and the door opened by that process

4.12 GetConf

Description
Returns information on the gateway

Syntax
```
bool GetConf        (string* stringaXML)
boolean GetConf     (StringBuffer stringaXML);
```

Parameters
stringaXML
[out] Receives the information on the gateway

Return Values
If the function succeds, the return value is true else false

4.13 Registry

Description

It records the component in the Loader

Syntax

```
bool Registry        (string aar, string* result, string* uuid )
boolean Registry     (String aar, StringBuffer result, StringBuffer uuid);
```

Parameters

aar

[in] points out the file that it describes the ASSIST component

result

[out] Receives the result of the command

uuid

[out] Receives the id of the recorded component

Return Values

If the function success, the return value is true else false

4.14 Unregistry

Description

It eliminates the component from the Loader

Syntax

```
bool Unregistry         (string uuid, string* result)
boolean Unregistry      (String uuid, StringBuffer result);
```

Parameters

uuid

[in] the id of the recorded component

result

[out] Receives the result of the command

Return Values

If the function success, the return value is true else false

4.15 UnregistryR

Description

It eliminates the component from the Loader and from the other known Loader.

Syntax

```
bool UnregistryR         (string uuid, string* result)
boolean UnregistryR      (String uuid, StringBuffer result);
```

Parameters

uuid

[in] the id of the recorded component

result

[out] Receives the result of the command

Return Values

If the function success, the return value is true else false

4.16 GetComponent

Description
It recovers the component if it is present in the Loader.
Syntax
```
bool GetComponent         (string uuid, string nomefile, string* result)
boolean GetComponent      (String uuid, String nomefile, StringBuffer result);
```

Parameters
uuid
[in] the id of the recorded component
nomefile
[in] name of the file where the component will be memorized
result
[out] Receives the result of the command

Return Values
If the function success, the return value is true else false

4.17 GetComponentR

Description
It recovers the description of the component if it is present in the Loader.
Syntax
```
bool GetComponentR         (string uuid, string nomefile, string* result)
boolean GetComponentR      (String uuid, String nomefile, StringBuffer result);
```

Parameters
uuid
[in] the id of the recorded component
nomefile
[in] name of the file where the component will be memorized
result
[out] Receives the result of the command

Return Values
If the function success, the return value is true else false

4.18 GetDescrComponent

Description
It recovers the description of the component if it is present in the Loader.
Syntax
```
bool GetDescrComponent         (string uuid, string nomefile, string* result)
boolean GetDescrComponent      (String uuid, String nomefile, StringBuffer result);
```

Parameters
uuid
[in] the id of the recorded component
nomefile
[in] name of the file where the description of the component will be memorized
result
[out] Receives the result of the command

Return Values
If the function success, the return value is true else false

4.19 GetDescrComponentR

Description
It recovers the description of the component if it is present in one of the known Loader
Syntax
```
bool GetDescrComponentR         (string uuid, string nomefile, string* result)
boolean GetDescrComponentR      (String uuid, String nomefile, StringBuffer result);
```

Parameters
uuid
[in] the id of the recorded component
nomefile
[in] name of the file where the description of the component will be memorized
result
[out] Receives the result of the command

Return Values
If the function success, the return value is true else false