ModCBroker: Programming Guide
Version 3.2.0
DocumentId:GradSoft-PR-e-05.12.2000-3.2.0
www.gradsoft.kiev.ua

October 15, 2008

Contents

1 Introduction 1

2 The main objects 2
- 2.1 Information about incoming http request . . . 2
  - 2.1.1 Information about names of files in HTTP file upload . 3
- 2.2 HTTPStream . . . 4
- 2.3 Handler . . . 5
- 2.4 Servlet . . . 7

3 Getting all together 10

4 Embedding CORBA Servlets into Web applications 10
- 4.1 Passing Query Parameters . . . 11
- 4.2 Filter Chains . . . 12
- 4.3 Authorization . . . 12

5 Scripting API 13
- 5.1 WebSh . . . 13

6 API for interaction with other Apache modules 14

7 IDL interfaces 15

8 Changes 21

1 Introduction

`ModCBroker` provides simple but powerful technique for building WEB applications and integrating CORBA application into WWW world. Work of `ModCBroker`, as any other application server engine, aims to translate incoming http requests to CORBA IIOP requests and provide to application developer programming interface (API) for input of http requests parameters and output of results to WWW browser of end-user.

I. e. general situation is shown on the next picture:

$$(\text{CORBA Server}) \leftarrow \frac{Intranet}{CORBA} \rightarrow (\text{cbroker}) \leftarrow \frac{Internet}{http} \rightarrow (\text{WWW browser})$$

Provided API is fully shown in section **IDL interfaces** 7. Roughly we can describe process of building WWW application with help of `cbroker` by next process:

1. Application developer implements CORBA interfaces HTTP::Servlet and HTTP::Handler.
2. Incoming http requests are authorized by HTTP::Servlet, then, if authorization was successful, Servlet creates Handler and passes it to ModCBroker.
3. Then ModCBroker calls method handle, of HTTP::Handler, where parameters are: information about request and HTTPStream type object .
4. HTTPStream provides API for output of information into client browser.

2 The main objects

2.1 Information about incoming http request

Are passed in structure `RequestInfo`

```
struct RequestInfo
{
  ClientInfo        client_info;
  ServerInfo        server_info;
  ParameterSequence parameters;
  ParameterSequence cookies;
  BinaryParameterSequence binary_parameters;
  string            method;
};
```

As we see, it includes:

- - information about client:

```
struct ClientInfo
{
  string ip;
  string hostname;
  string user_name;
};
```

where

ip – IP of client machine (or his proxy).

hostname – Domain name of client (or his proxy) Internet host.

user_name – User name. If servlet does not require authorization, than string "unknown" is passed.

- - information about WWW server. (note, that few WWW servers with ModCBroker can address to the same CORBA servlet).

```
struct ServerInfo
{
  string hostname;
  short  port;
};
```

- - information about text GET & PUT parameters of request.

```
struct Parameter
{
  string  name;
  string value;
};
typedef sequence<Parameter>  ParameterSequence;
```

Note, that GET and PUT parameters are identical from CORBA servlet point of view.

- - information about cookies

- - information about binary parameters of POST and PUT requests.

```
struct BinaryParameter
{
  string  name;
  CORBA::OctetSeq value;
};
typedef sequence<BinaryParameter>  BinaryParameterSequence;
```

It is possible to pass binary data through http to CORBA server: for example organize file upload form html form.

- - method: HTTP method as string: one of "PUT", "GET", "OPTIONS" and so on. Why we decided to use string instead of enum: to allow easy extensions of HTTP protocol "in Apache way". For example, by handling "DELETE" methods you can easy implement WebDav protocol functionality.

2.1.1 Information about names of files in HTTP file upload

ModCbroker can handle file upload HTTP extension, in such case information about uploaded files is stored in `binary_parameters` field of `RequestInfo`.

`binary_parameters.name` contains `<input .. >` HTML tag name in file upload form, `value` - content of uploaded file.

For example, suppose that our file upload form looks as follows:

```
<INPUT TYPE=FILE NAME="pics">
```

then name of input tag is "pics", so according component in `binary_parameters` will have name `"pics"`.

Full file name can be extracted from Parameters where name of parameter is `name+"_filename"`. In this case name of parameter is `"pics_filename"`.

2.2 HTTPStream

HTTPStream is CORBA Object, which provides API for output into client browser, by exporting Apache [?] API for these purposes

For example, following code fragment is used to output HTML page "Hello, world":

```
httpStream.set_content_type("text/html");
httpStream.send_http_header();
httpStream.puts("<HTML><HEAD><TITLE>Hello, World</TITLE><HEAD>"
                "<BODY><CENTER>Hello, World</CENTER></BODY></HTML>");
```

More detailed form:

```
interface HTTPStream
{
  void set_content_type(in string content_type);
  void set_http_header(in string name, in string value);

  void set_cookie(in string name, in string value, int long expire_time);

  void send_http_header();

  void send_http_header_ex(in ReplyHeaderInfo headerInfo);
```

```
  string get_http_header(in string name);

  void puts(in string str)
              raises(StreamClosedException);

  unsigned long send_buffer(in CORBA::OctetSeq buffer)
              raises(StreamClosedException);

  void flush()
              raises(StreamClosedException);

};
```

- `void set_content_type(in string content_type);` This method sets content type of output html. Argument is a string of mime type (for example: "text/html" or "image/gif")
- `void set_http_header (in string name, in string value);` This function sets variable in header of HTTP reply
- `set_cookie` – sets cookie, which will be passed to client. The meaning of parameters is obvious. Expire time must be in seconds. During next request to servlet, this cookie will be read and passed to servlet in request parameters.
- `void send_http_header()` – send header of http-reply to client. Functions, which work with output http header (`set_content_type`, `set_http_header`, `set_cookie`) must be called before `send_http_header`; function which work with output http body must be called accordingly after.

  `send_http_header_ex` - is composition of `set_content_type`, `set_http_header` and `send_http_header` methods. It accept structure `ReplyHeaderInfo` with all information about header of request output:

  ```
  struct ReplyHeaderInfo
  {
   string content_type;
   ParameterSequence fields;
   ParameterSequence cookies;
   unsigned long     cookie_expire_time; // in seconds.
  };
  ```

  Note, that all cookies, passed to `send_http_header_ex` have identical lifetime - `cookie_expire_time`.

  Usage of `send_http_header_ex` can minimize number of round-trips between Servlet and ModCbroker during request processing, so we recommend to use this method for overloaded sites.

- string get_http_header(in string name); - returns input header of request with name name. Note, that set(get(x)) != x, i. e. get_http_reader looks in input headers, while set_http_headers set output request headers.
- void puts (in string str) – outputs string to client in output request body.

  If connection with client is dropped for some reason this function, as others, is working with body of reply, throws StreamClosedException.
- unsigned long send_buffer(in sequence<octet> buffer) – sends binary sequence of bytes to client. This function can be used for transferring of files and images.
- void flush() – force passing data to client, by cleaning of internal buffer of WWW server.

2.3 Handler

Ok, now we know how to use API of http output. The next question: where to use the it ? – in application programmer implementation of Handler interface.

```
interface RequestHandler
{
  void  handle(in RequestInfo request_info, in HTTPStream stream)
                      raises(ExternalException,
                             StreamClosedException,
                             Redirect);
  void  destroy();
};
```

Developer must implement it as CORBA object, and put processing of request into method handle

- handle process incoming request.
  - request_info – information about request.
  - stream – HTTP stream, which provides API for working with client WWW browser.

  This method can throw exception "ExternalException", in such case exception reason would be passed to client and logged in log file of Apache.

  ```
  exception ExternalException
  {
    short http_code;
    string reason;
  };
  ```

Note, that exception StreamClosedExcepthion is thrown from API HTTP-Stream.

It is put in declaration of Handler interface, for next reasons: if application programmer does not catch one, than www server must do all necessary operation of cleaning requests itself.

At last, `Redirect` exception is passed to `Apache` for internal redirection of apache request - it is useful for integration of `ModCbroker` with web scripting.

```
exception Redirect
{
 string url;
 ParameterSequence parameters;
};
```

Note, that `ModCbroker` transforms `Redirect` exception to get request.

- `destroy` Deactivate CORBA object, according to ORB rules. (For more detailed description of lifecycle rules for CORBA objects, look at [?], [?], [?]). Note, that if handler is static, that destroy can do nothing. (i. e. be empty)

So, now we can write Handler, for output of HelloWorld:

```
class HelloWorldHandler: POA_HTTP::RequestHandler
{
   PortableServer::POA_var poa_;
 public:
   HelloWorldHandler(PortableServer::POA_ptr poa)
     :poa_(PortableServer::_duplicate(poa)
   {}

   void handle(const RequestInfo& request_info, HTTPStream_ptr httpStream)
   {
     httpStream.set_content_type("text/html");
     httpStream.send_http_header();
     httpStream.puts("<HTML><HEAD><TITLE>Hello, World</TITLE><HEAD>"
                  "<BODY><CENTER>Hello, World</CENTER></BODY></HTML>");
   }

   void destroy()
   {
     PortableServer::ObjectId_var oid = poa_->servant_to_id(this);
     poa_->deactivate_object(oid);
     _remove_ref();
   }
```

```
};
```

2.4 Servlet

The next point is a "Servlet" interface. This interface also must be implemented by application programmer, the main tasks of Servlet are:

- Authorization: servlet must decide: must request be authorized, and if yes – implement authorization method.
- Generation of handlers. (i. e. in terms of now popular design patterns, Servlet is a factory for handlers.). Note, that in case of static set of handlers, you shouldn't process creating/destroying of objects in language implementation level. (i. e. lifetime of handler can be equal of lifetime of servlet, Servlet::createHandler return same reference; Handler::destroy can do nothing).

Interface description:

```
interface Servlet
{

   RequestHandler create_handler(in string handler_name,
                                 in ClientInfo client_info,
                                 in ServerInfo server_info,
                                 in string passwd)
                                     raises(NoSuchHandlerException,
                                            RequireAuthInfo,
                                            AccessDenied,
                                            ExternalException,
                                            Redirect);
```

Method createHandler authorizes owner of request and creates handler for request; then cbroker will call method handle in it, `handle` after – destroys handler. If username or password can't pass authorization, createHandler raises RequireAuthInfo exception. In other case if authorization failed createHandler raises AccessDenied exception.

Let's write servlet in C++, which starts our HelloWorld handler in answer to the name "hello". And let's suppose that we allow execute this handler only for user with name "admin" and password "qq888ikd930" from host 200.200.220.10

```
class HelloWorldServlet: public POA_HTTP::Servlet
{
 PortableServer::POA_var poa_;
public:
```

```
 HelloWorldServlet(PortableServer::POA_ptr poa)
  :poa_(PortableServer::_duplicate(poa))
 {}

 HTTP::RequestHandler_ptr createHandler(in string handlerName
                                        const HTTP::ClientInfo& clientInfo,
                                        const HTTP::ServerInfo& serverInfo,
                                        const char* passwd)
 {
  if (strcmp(handlerName,"hello")!=0) throw NoSuchHandlerException();

  // allow access only from host 200.200.220.10 to user admin with
  // password qq888ikd930
  if (strcmp(clientInfo.user_name,"admin")!=0) throw RequireAuthInfo();
  if (strcmp(clientInfo.ip,"200.200.220.10")!=0) throw AccessDenied();
  if (strcmp(passwd,"qq888ikd930")!=0) throw RequireAuthInfo();
  if (strcmp(handlerName,"hello")!=0) throw NoSuchHandlerException();

  HelloWorldHandler* handler_impl = new HelloWorldHandler(poa_);
  HTTP::RequestHandler_var retval = handler_impl->_this();
  return retval._retn();
 }

};
```

You can use

For example, let's write Servlet which does only authorization of users in database and redirect user to his personal homepage.

```
class CheckServlet: public POA_HTTP::Servlet
{
 UAKGQuery2::QueryManager_var queryManager_;
 public:

   CheckServlet(UAKGQuery2::QueryManager_ptr queryManager)
    :queryManager(UAKGQuery2::QueryManager::_duplicate(queryManager))
   {}

   HTTP::RequestHandler_ptr createHandler(in string handlerName,
                                        const HTTP::ClientInfo& clientInfo,
                                        const HTTP::ServerInfo& serverInfo,
                                        const char* passwd)
   {
    if (!strcmp(clientInfo.username,"unknown")) {
      throw RequireAuthInfo();
    }
```

```
    UAKQUery2::RecordSet_var params=UAKGQuery2::RecordSet::create(2,1);
    params.setColumnName(0,"login");
    params.setColumnName(1,"passwd");
    params.setString(0,0,clientInfo.username.in());
    params.setString(0,1,passwd);
    UAKQUery2::RecordSet_var rs = queryManager_->evaluate(
                               "select home_url from users where "
                                         "login = :login "
                                         " and "
                                         "password = :password "
                              );
    if (rs.getNRows()==0) {
           throw AccessDenied();
    }
    String_var home_url=rs.getAsString(0,0);
    Redirect redirect;
    redirect.url=CORBA::string_dup(home_url.in());
    throw Redirect();
   }

};
```

3 Getting all together

Last question: In case we implement all, how `cbroker` will find servlet and handler name by URL?

Answer: with the help of CORBA Name Service [?]. I. e. if path part of request URL looks like:

$$http://host.com/cbroker-dir/ServletName/HandlerName$$

(where cbroker-dir – name of virtual directory, set in WWW server config, then `ModCBroker` will find servlet in default NamingService with name HTTPServ/ServletName.

Servlet developer must bind servlet with name HTTPServ/ServletName during initialization of Servlet. You can find examples in directory demo in source distribution of `ModCBroker`

At last, program which uses cbroker API must be compiled with cbroker client library (clcbroker for C++; for java you can create clcbroker.jar), and process of WWW server tuning is described in [?]

4 Embedding CORBA Servlets into Web applications

In some cases we need separate generation of content and presentation. For example, some part of web application (data delivery or back-end functionality)

must be CORBA Servlets, but design must be controlled by web-designer, not by programmer.

Such separation can be implemented in two ways: embedding template processing and scripting on servlet side (for example, providing template processing as CASL service) and on Apache side. For relative small applications last way is quite comfortable.

So, we added support of embedding cbroker calls into web pages and apache script modules: cbroker provides filters to process presentation, and provide API for call of cbroker servlets from script languages. [1].

Using CBROKER filter allows to embed calls of ModCbroker servlets into web pages: expression like

```
<?cbroker ServletName HandlerName p1="v1" ... pN="vN" ?>
```

is transformed to result of call of handler `HandlerName` in servlet `ServletName` with parameters $\{p_i, v_i\}$. During such call `method` field in structure `RequestInfo` is set to string `'CBROKER-INCLUDE'`.

For enabling such processing we must adjust processing of HTML text by Apache output filter with name `CBROKER`.

For example, if we have servlet for output of some information about enterprise, and handler which outputs list of all internal phones of employees is named `NSI/emp`.

With tradition programming technique we must call application programmer to modify appropriative servlet during each change of site design. With `ModCbroker-3.1.0` we can do next:

- Write template in file with extension `.cbhtm`:

```
<HTML>
<HEAD> <TITLE> List of employees </TITLE> </HEAD>
<BODY>
 <img src="images/OurGreatImageOnTop.gif" /><br></br>
 <?cbroker NSI Emp ?>
 <img src="images/OurGreatImageOnBottom.gif" /><br></br>
</BODY>
</HTML>
```

- tell Apache to apply CBROKER filter to files with extension `cbhtml`, by adding next command to Apache configuration file:

```
AddOutputFilterByType CBROKER cbhtml
```

That's all: now programmer can work only with semantics, designer — with design.

[1] when script interpreter implemented as Apache module

4.1 Passing Query Parameters

For example we have handler, which searches employee by name and output detailed information. Let it be `NSI/emp_search`. For example, home page of Jon Scott[2] includes next block:

```
<?cbroker NSI emp_search name="jon" family="scott" ?>
```

It's good for static pages, but what we must do, when we want to design page with cbroker includes which outputs information for employee, choosen by user in form ? For this we have special syntax:

```
<?cbroker NSI emp_search passRequestParameters="1" ?>
```

We can pass parameters to servlet from request query and from text at the same time:

```
<?cbroker NSI emp_search family="scott" passRequestParameters="1" ?>
```

In this case, when we have the same name of parameter in request and in text, only parameter from text is passed to servlet. Why — for safety: to have the possibility to write SQL text and not to be afraid of substitution.

And last — we have few different methods of parameter passing:

- `passRequestParameters` - pass all parameters from request.
- `passQueryArgs` – pass parameters only for GET request method.
- `passPostAndPutParameters` – use parameters only from POST or PUT client stream.

4.2 Filter Chains

Apache filters can be combined: for example we can use `mod_cbroker` together with `mod_include`:

- set such combination of filters in Apache configuration:

  ```
  AddOutputFilterByType CBROKER;INCLUDE  .cbshtml
  ```

  this directive says, that files with type `cbshtml` will be processed at first with `CBROKER` filter, than with `INCLUDE` .

- use together `mod_include` and `mod_cbroker` tags, for example:

  ```
  <HTML>
  <HEAD> <TITLE> List of employees </TITLE> </HEAD>
  <BODY>
   <--#include virtual "/topOfOurPage/" --><br></br>
  ```

[2] with password tiger ;)

```
 <?cbroker NSI Emp ?>
 <--#include virtual "/bottomOfOurPage/" --><br></br>
</BODY>
</HTML>
```

Of course we can use other filters: for example cbroker output can be processed by `xslt` filter, or we can generate cbroker calls from PHP. Hmm.. for PHP better use cbroker scripting API, because combination of text substitution and programming language looks not very natural.

4.3 Authorization

When we embed `cbroker` calls into html pages, we need to implement some authorization routine: server must know name of user before outputting the page. We have directive `CbrokerAddAuthByLocation` for this purpose. Syntax:

```
CbrokerAddAuthByLocation  <location> <servlet> <handler>
```

This directive tells Apache to call authorization handler (defined by `<servlet>` and `<handler>` parameters) during access to any resource situated inside location `<location>`.

For example, directive

```
CbrokerAddAuthByLocation /cbf Billing Auth
```

means that access to all files, situated in virtual directory `/cbf` and in all subdirectories is determined with help of `auth` handler in `Billing` servlet. Protocol of interaction is very simple: servlet during call of authentication handler can throw exception `RequireAuthentification`. Only one difference from normal handler : in successful case we don't need return value, so we can return NIL.

Example:

```
class HelloWorldServlet: public POA_HTTP::Servlet
{
 PortableServer::POA_var poa_;
public:
 HelloWorldServlet(PortableServer::POA_ptr poa)
  :poa_(PortableServer::_duplicate(poa))
 {}

 HTTP::RequestHandler_ptr create_handler(in string handlerName
                                         const HTTP::ClientInfo& clientInfo,
                                         const HTTP::ServerInfo& serverInfo,
                                         const char* passwd)
 {
  if (strcmp(handlerName,"auth")==0) {
    // only for admin from host 200.200.220.10 and with password qq888ikd930
```

```
    if (strcmp(clientInfo.user_name,"admin")!=0) throw RequireAuthInfo();
    if (strcmp(clientInfo.ip,"200.200.220.10")!=0) throw AccessDenied();
    if (strcmp(passwd,"qq888ikd930")!=0) throw RequireAuthInfo();

    return HTTP::RequestHandler::_nil();
  }
 }

};
```

We must also tell Apache to apply authorization procedure to target location. This is described in Administration guide, so read it before using this directive.

5 Scripting API

5.1 WebSh

You can call `ModCbroker` servlets from `websh` [?] scripts. Syntax:

```
cbroker servletNamr handlerName {param-list}
```

In parameters we must have sequence of pairs name/value. For example:

```
[cbroker Billing AccountState { username $username userid $userid } ]
```

Before using `cbroker` extension we must load one from shared library `libwebshcbroker.so`
For example, let's call our `Hello, World` servlet:

```
#
# Demo for interactuion of ModCbroker with Tcl
#
web::initializer{
  # load cbroker interaction library
  load /usr/local/lib/libweshcbroker83.so.1
}

web::command default {

 web::put {
   "<HTML><HEAD>  Hello World demo </HEAD>"
   "<BODY>"
   "<center>"
   "    [cbroker Hello World {} ] "
   "</center>"
   "</BODY>"
   "</HTML>"
 }
```

```
}

web::dispatch
```

6 API for interaction with other Apache modules

If you want to call cbroker from other Apache2 module (for example, for embedding cbroker into your favorite scripting language), then cbroker provides you with next function:

```
APR_DECLARE_OPTIONAL_FN(int,ap_cbroker_call,
            (request_r* where,
             const char* servletName,
             const char* handlerName,
             const char*  method,
             apr_table_t* parameters,
             bool useOutputBuffer,
             char** outputBuffer,
             apr_size_t* outputLen
            ))
```

Technique for accessing optional functions is described in [?].

7 IDL interfaces

```
#ifndef __HTTP_IDL
#define __HTTP_IDL
/*
 * Interface for CORBA http servlets.
 * (C) GradSoft 2000, 2001, 2002
 * http://www.gradsoft.com.ua
 * (C) Ruslan Shevchenko <Ruslan@Shevchenko.Kiev.UA>
 *  1999, 2000, 2001, 2002
 * $Id: ProgrammingGuide_eng.tex,v 1.18 2007-06-14 11:47:14 rssh Exp $
 */


#pragma prefix "gradsoft.kiev.ua"

/**
 * Module for CORBA http servlets.
 **/
module HTTP {
```

```
///

///
typedef sequence<octet> HTTPOctSeq;
///
typedef sequence<string> HTTPStrSeq;

/**
 * used for passing binary parameters from
 * users forms
 **/
struct BinaryParameter
{
  ///
  string  name;
  ///
  HTTPOctSeq value;
};
///



/**
 * used for passing parameters from
 * users forms
 **/
struct Parameter
{
  ///
  string  name;
  ///
  string value;
};


///
typedef sequence<Parameter>  ParameterSequence;
///
typedef sequence<BinaryParameter>  BinaryParameterSequence;

/**
 * what we know about client:
 **/
struct ClientInfo
{
```

```
  /**
   * IP adress of client
   **/
  string ip;
  /**
   * hostname of client
   **/
  string hostname;
  /**
   * user name of client.
   * (or 'unknown', if handler does not require authorization)
   **/
  string user_name;
};

/**
 * what we know about web server
 **/
struct ServerInfo
{
  ///
  string hostname;
  ///
  short  port;
};

/**
 * this structure describes user request
 **/
struct RequestInfo
{
  ///
  ClientInfo        client_info;
  ///
  ServerInfo        server_info;
  ///
  ParameterSequence parameters;
  ///
  ParameterSequence cookies;
  ///
  BinaryParameterSequence binary_parameters;
  ///
  string            method;
};
```

```
/**
 *
 **/
struct ReplyHeaderInfo
{
  string            content_type;
  ParameterSequence fields;
  ParameterSequence cookies;
  unsigned long     cookie_expire_time;
};

/**
 * Can be raised by user servlets code.
 **/
exception ExternalException
{
  /// http code, which we want return to client
  short http_code;
  /// string description of error.
  string reason;
};

/**
 * raised by web server, when user output stream is cancelled
 **/
exception StreamClosedException {};

/**
 * raised by create_handler(), when user login and password required
 **/
exception RequireAuthInfo {};

/**
 * raised by create_handler(), when ClientInfo::ip or ClientInfo::hostname
 * or ServerInfo::hostname or ServerInfo::port has no access to this
 * servlet or handler
 **/
exception AccessDenied {};

/**
 * raised in createHandler() or in handle() to redirect request
 **/
exception Redirect {
    // url to redirect
    string url;
```

```
    // parameters for uri
    ParameterSequence parameters;
};

/**
 * Servel communicate with web server (and users web browser)
 * via this interface, during handling of request.
 **/
interface HTTPStream
{
  ///

  /**
   * set content type of servlet output.
   *@precondition
   *  it must be done before call of send_http_header
   **/
  void set_content_type(in string content_type);

  /**
   * set additional http header name-value pair
   *@precondition
   *  it must be done before call of send_http_header
   **/
  void set_http_header(in string name, in string value);


  /**
   * get http header value for requested name
   * return value of http header in request.
   **/
  string get_http_header(in string name);

  /**
   * set cookie in output client header.
   *@precondition
   *  it must be done before call of send_http_header
   **/
  void set_cookie(in string name, in string value,
                  in unsigned long expire_time );

  ///
  void send_http_header();

  ///
  void send_http_header_ex(in ReplyHeaderInfo header_info);
```

```
  /**
   * send string to user browser
   **/
  void puts(in string str)
              raises(StreamClosedException);

  /**
   * send sequence of octet to user browser.
   **/
  unsigned long send_buffer(in HTTPOctSeq buffer)
              raises(StreamClosedException);

  /**
   * flush output stream.
   **/
  void flush()
              raises(StreamClosedException);

};

/**
 * Application programmer must provide implementation of this
 * interface.
 **/
interface RequestHandler
{

  ///

  /**
   * handle request and output results to HTTPStream
   **/
  void  handle(in RequestInfo request_info, in HTTPStream stream)
             raises(ExternalException, StreamClosedException, Redirect);

  /**
   * destroy request handler. This method is
   * called by servlet engine after processing of request.
   * programmer must not call this method directly.
   **/
  void  destroy();
};

/**
 * throwed by servlet when we try to call unexistent handler
```

```
    **/
   exception NoSuchHandlerException {};


   /**
    * Application programmer must provide implementation
    * for this interface and bind it to "HTTPServ/Name" in
    * Naming Service
    **/
   interface Servlet
   {
      /**
       * create handler for request.
       *@exception NoSuchHandlerException when we try to call
       *            unexistent handler
       *@exception RequireAuthInfo  when Servelt require filled client name
       *            and client password structure, but it is not passed with
       *            this call (and ModCbroker popup auth window on next step)
       *@exception AccessDenied when access for this user is denied
       *@exception ExternalException can be throwed by servlet.
       **/
      RequestHandler create_handler(in string handler_name,
                                    in ClientInfo client_info,
                                    in ServerInfo server_info,
                                    in string passwd)
                                       raises(NoSuchHandlerException,
                                              RequireAuthInfo,
                                              AccessDenied,
                                              ExternalException,
                                              Redirect
                                             );
   };


};

#endif
```

8 Changes

- 13.06.2007 - updated for `mod_cbroker-3.2.0`
- 06.02.2003 - updated for `mod_cbroker-3.1.0`
- 21.05.2002 - reviewed for `mod_cbroker-3.0`

- 26.02.2002 - updated for `mod_cbroker-2.0.5`
- 16.01.2002 - review.
- 27.05.2001 - last touch
- 24.04.2001 - updated for `mod_cbroker-2.x`
- 17.03.2001 - review.
- 09.02.2001 - correction of English grammar.
- 30.01.2001 - review, added formal documentation set attributes.
- 14.05.2000 - created.

References

[1] Ryan Bloom. Apache modules. *ONLAMP*, 2001. http://www.onlamp.com/pub/a/apache/2001/09/27/apache_2.html.

[2] Object Management Group, editor. *Common Object Services Specification*, chapter 3. Naming. OMG, 1997. formal/97-12-10.

[3] Object Management Group, editor. *The Common Object Request Broker: Architecture & Specification*, chapter 11. Portable Object Adapter. OMG, 1999. formal/99-10-07.

[4] Michi Henning and Steve Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley, 1999. ISBN 0201379279.

[5] Ben Laurie and Peter Laurie. *Apache. The Definitive Guide.* O'Reilly, 1997. ISBN 1-56592-250-6.

[6] Apache Software Foundation NetCetera AG. *websh home page*, 2000-2002. http://tcl.apache.org/websh.

[7] Object Oriented Concepts, inc. *ORBacus for C++ and Java.*

[8] Julia Prokopenko Ruslan Shevchenko, Alexandr Yanovec. *mod_cbroker: Administration Guide*, 2000-2001. GradSoft-CBroker-e-AG503-79-10.05.2000-2.0.1.