

Communication with the IPAS-server via clsIpas-library

Contents

Information about the System	3
IPAS Server.....	3
ClsIpas Library.....	3
RULE.XML.....	3
UI above the clsIpas Library	3
Communication protocol	4
XML	4
Heading	4
The heading always contains 8 byts and has the following format:	4
Heading for GZIP format	4
Connection rules.....	4
Description of the clsIpas library	5
Library parameters	5
Socket connection parameters	5
Parameters controlling GZIP manner.....	5
Parameters determining place and name of instruction command files.....	5
Instruction command counters.....	5
Parameters for instruction commands preparation.....	5
Parameters for incoming data.....	5
Parameters XML data and index	6
Data for character conversion	6
Library methods	6
Description of the IpasElementFunction Library	7
General description	7
Utility.....	7
Implementation	7
Example	7
Generates the output:	7
Description of added function in the IpasAddFunction Library	8
ArrMultiArrayToSingleArray Function.....	8
arrQuery Function.....	8
ArrFromXML Function.....	8
arrIpasFromXML Function.....	8
Dynamic generating of instruction commands and the clsIpas library	9
Basic about generating.....	9
Configuration.....	9
Names, default values and explanation of the single configurations are following:.....	9
Principle of library and instruction command creation.....	10
Instruction commands	11
Error codes of the clsIpas library	11
Value list of code key, their meaning and the meaning of the second optional key:..	11

Instruction command list.....	11
The Ipas server currently supports following commands available for partners or sub registrars:.....	11
Description of important commands	14
Login	14
GetDomainInfo	14
CheckDomain	14
Query.....	14
Examples	15
Demonstration of Login command via clsIpas library	15
Changes	16
Version 2.2.1 (20051012).....	16
Correction of the Query ‘clsIpas.tpl’ class.....	16
Version 2.2.0 (20050727).....	16
Change of ‘generate.php’	16
Version 2.1.0 (20050722)	16
Change of ‘clsIpas.tpl’	16
Version 2.0.1 (20050712)	16
Change of ‘clsIpas.tpl’	16
Version 2.0.0 (20050623).....	16
Change of ‘clsIpas.tpl’	16
Change of ‘generate.php’	16
Version 1.1.2 (20050510)	16
Correction of ‘generate.php’	16
Version 1.1.1 (20050415)	17
Correction of ‘generate.php’	17
Version 1.1.0 (20050331)	17
Correction of ‘generate.php’	17
Version 1.0.3 (20041026)	17
Correction of ‘generate.php’	17
Version 1.0.2 (20041025)	17
Correction	17
Version 1.0.1 (20040924)	17
IPAS (20040927)	17
Added	17
Changes.....	17
Correction.....	18

Information about the System

IPAS Server

The IPAS Server is the operating core for registration of CZ, generic and other domains. In the future IPAS will also become the core for registration and administration of webhosting and other services provided by Ignum Company. IPAS also contains tools for providing services of other companies through Ignum (e.g. partners and sub registrars), who can at will communicate with the server through a communication interface described below.

Now the server contains instruction commands for complete registration of CZ, generic and other domains. Further there are instruction commands for controlling of billing, ordering, credit and user's personal set-up, and user's account administration.

Clslpas Library

clslpas Library was developed because of the demand of partners and Ignum's sub registrars to communicate with the IPAS server.

Library development language is PHP in its current version 4.3.8. Further development will be administrated in PHP 5 and newer, more advanced in library programming.

This library results from Ignum,s internal library, request_xml, which was offered as the first communication agent with the IPAS server. This library was considered deficient mainly in the area of actualisation of instruction commands, linkings and definitions, which are to be followed during communication, and thereupon clslpas Library was developed. Clslpas is distributed in source code through its template. That is the headstone for the clslpas library dynamic generation and instruction commands for communication via set of rules rule.xml.

RULE.XML

The file rule.xml contains definitions and rules for communication with the IPAS server. Due to this file it is possible to create a valid instruction command code and to treat all errors already on the client's part. Thereby the communication necessity with the IPAS is economized.

A generating script is concerned with the rules set parsing. From defined rules this generating script is able to create single instruction commands with predefined return codes and entry data format. It is also possible to gain the instruction command format from the file in another way.

The file itself contains notes about rule type. It is possible to trace up simple principles from the generating script.

UI above the clslpas Library

The clslpas library should above all serve as the headstone of user's interfaces communicating with the IPAS server.

Communication protocol

XML

The communication protocol of the IPAS server is the XML language. The entire communication proceeds in UTF-8 language according to the xml-1.0 definition.

Entire instruction command definitions can be gained through the rule.xml file, which contains individual GroupElements and Elements. It is necessary to be aware of the fact that the XML language is CaseSensitive.

It is also needful to be aware of the translation necessity of special HTML characters not allowed in the xml language. The substitution is to be carried out by HTML entities.

The clsIpas library is ready to receive data in predefined code page and not translated into html entities. Both are done during preparation of XML instruction command, so there would not be any disorganisation of the xml-1.0 specification during communication and the server would not refuse the instruction command.

Heading

Every request, or every connection with the server has to be started with a heading. This heading is controlled by the server and in case of locating any error the communication is refused.

The heading always contains 8 bytes and has the following format:

- DWORD m_dwSize; - request size without heading
- BYTE m_byOrder; - request sequence in open connection)
- BYTE m_byFormat;- request format (original format or gzip)
- WORD m_wSum; - heading simple sum

Heading for GZIP format

In case you are using GZIP connection, it is necessary to add a heading containing the size of an unpacked data summary in front of every outgoing instruction command summary.

Connection rules

During one connection with the IPAS server it is necessary to pay attention to the Login instruction command execution. By this instruction command you are authorising towards the server and thereby you authorise the whole connection up to it's end, to the LogOut instruction command or up to a new Login. The Login instruction command is possible in several types at once. As a common user (sub registrar) you log on with the 'Account' type, as a partner then with the 'Partner' type. The 'Admin' type is only for the system administrator.

Description of the clslpas library

Because PHP 4 doesn't support the encapsulation of parameters or library method, it is possible to use all these parameters or methods. The library indeed is written in a way so that it is necessary to use only some of the basic ones servicing the entire library. This is easy to elicit from the enclosed communication examples.

Library parameters

Socket connection parameters

1| \$hCoreSocket = False; Core connection indicator.

Parameters controlling GZIP manner

- **\$bGZData = false**; Defines default compression manner
- **\$iGZLevel = 9**; Determines compression level (1 – minimal compression, 9 – maximal compression)
- **\$iGZMoreThan = 1024**; Gzipes only instruction commands greater than the set value (in bytes without heading)
- **\$aGZAllways = Array(,Login')**; Defines the area of always gzipping instruction commands independently of the rate in \$bGZData
- **\$bGZDataNow = false**; Is instrumental to compression status deposition for the actually outgoing request

Parameters determining place and name of instruction command files

- **\$sRequestFcePrefix = ,<%sRequestFcePrefix%>'**; Function prefix of single requests (is instrumental to avoid conflicts with functions of your own system)
- **\$sRequestFileExtend = ,<%sPostFixRequestTmp%>'**; Extension of files containing instruction commands and their definitions
- **\$mExternalFunctionsPath = ,,**; Parameter with relative address for placing files with instruction command definition

Instruction command counters

- **\$iRequestCounter = 0**; Counter of instruction commands in the request | \$iRequestSendCounter = 0; Counter of instruction commands in an open connection.

Parameters for instruction commands preparation

- **\$aRequestData = Array()**; Parameter for the requests entering data.
- **\$aRequestOut = Array()**; Parameter with resulting XML request.
- **\$sRequestCache = ,,**; Cache of the instruction command body

Parameters for incoming data

- **\$sRequestResult = ,,**; Resulting XML answer from the core.

Parameters XML data and index

- **\$aParseXmlValues = Array();** XML parser output value field.
- **\$aParseXmlIndexes = Array();** XML parsers index field.

Data for character conversion

- **\$aCharsetBoth = Array();** Help field for code pages translation.
- **\$aCharsetIn = Array(<%sCharsetIn%>);** A field with problematic characters in inner code page
- **\$aCharsetOut = Array(<%sCharsetOut%>);** A field with problematic characters in UTF-8

Library methods

- **__construct()** – Library constructor.
- **__destruct()** – Library destructor.
- **strCreateRequest()** – creates a \$aRequestOut field from actual data in \$aRequestData. If there wasn't an error, the return will be a string with a XML request. In case of an error there will be a field with error codes of single instruction command elements on the output.
- **unCallExternalFunction(\$sRequestName, array \$aInRequestData = „,“)** – Finds out and exclaims the function with the definition of the given instruction command. If the instruction command wasn't found or an error occurs, a false returns. Otherwise a release value of the exclaimed function is returned.
- **arrayConvertInData(&\$aInData)** – is a recursive translator of incoming data into right htmlentities. **strConvertData(\$sInData, \$sOrder)** – converts data between code pages with the help of problematic characters defined fields.
- **strConvertRequest()** – puts data together from the parameter \$aRequestOut into the resulting XML string.
- **voidParseXml()** – fills data and index fields with the parsed XML core answers
- **strCreateHeader()** – creates a request heading
- **Command()** – sends the request to the server and processes the answer
- **OpenConnect()** – opens the connection to the server
- **CloseConnect()** – closes the connection to the server
- **ResetSession()** – resets the actual connection, eventually creates a new one

Description of the IpasElementFunction Library

General description

The IpasElementFunction class is there for the possibility to set the given request element as an inner Ipas function. Inner Ipas function is above all instrumental to the translation of values such as Domain Name or RRID to values such as ID, which make the programmer's job easier and economizes the usage of Query instruction command.

Utility

If you enter an object type IpasElementFunction as an element value, the element will be processed in a slightly different way. Such an element will be created with the attribute function="1" and a value representing classical exclaim of the given function. The Ipas itself then first carries out the request for function exclaim, then places its output as the given element and first then it processes the whole instruction command in the common way.

Implementation

The IpasElementFunction class contains a list of functions being currently implemented in Ipas. At the same time it contains a list and type of attributes being required by the given function. Attribute types are only divided in Integer (I) and String (S), which differ by quotation marks usage by function exclaim generating (and also contents of the given element). In case you create an object of this type and enter a wrong function name, an IpasException will be evoked by generating it's output value (by usage in Request).

Example

```
$oIpas = new clsIpas();
$oQueryID = new IpasElementFunction('QueryCodeToID', array('Code' => '45'));
$aReqData[0]['name'] = 'Query';
$aReqData[0]['data'] = array(
, ID' => $oQueryID,
, Params' => array('ORDER' => 'date'));
$oIpas->aRequestData = $aReqData;
$sRequest = $oIpas->strCreateRequest();

print_r($sRequest);
```

Generates the output:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Commands>
  <Command ID="0">
    <Query>
      <ID function="1">QueryCodeToID(,48')</ID>
      <Params>
        <ORDER>date</ORDER>
      </Params>
    </Query>
  </Command>
</Commands>
```

Description of added function in the IpasAddFunction Library

ArrMultiArrayToSingleArray Function

Is instrumental to generate a more-dimensional field to a one-dimensional field. More dimensions are then read as a stop in the key of the given field.

The input is a field given over by a value. The output is a modified field.

arrQuery Function

Is instrumental to exclaim the Query instruction command via clsIpas library. The output is a simple and already pared value field that the Ipas returns in XML.

The input is a clsIpas communication library, via which the instruction command is carried out, Query instruction command code and the field of other parameters to be instituted into Query.

In this function it is possible to see the usage of IpasElementFunction, which is here instrumental to translate QueryCode into QueryID.

ArrFromXML Function

Is a function, which makes it possible to generate two fields emergent after XML, paring via an inner PHP parser, to a more-dimension field.

The input is an index and value field emerging after parsing (in clsIpas it is aParseXmlIndexes and aParseXmlValues) and the name of the tag, from which the output should be carried out.

The tag name is instrumental to define the „immersion“ into XML.

The output is a more-dimension field.

This function ignores the attribute values of single Xml output elements. The attributes are actually only used by the CheckDomain instruction command.

arrIpasFromXML Function

This function is only an alias to the arrFromXML function. However, her input values are not index and value fields but the clsIpas library itself. The function itself then only exclaims arrFromXML with the given fields from the delivered library.

The output is identical.

Dynamic generating of instruction commands and the clsIpas library

Basic about generating

To be able to easily add instruction commands, change their validation values and level down the number of errors and other questions, we created the file rule.xml, which contains a large number of there rules and scripts that can be parsed by this file and single instruction commands can be created.

The script for parsing the rules file is called generate.php and again is freely available in the source code. With regard to the clsIpas library developing language it is also created in PHP. Enclosed to the script there is a generate.var file, which contains the configuration for instrument command generating and the library.

Configuration

The parameters in the generate.var file directly influence the name, code page, location and the function names of single instrument commands.

Names, default values and explanation of the single configurations are following:

- **\$sFileRule = ,./rule.xml'**; - file with rules under which single instruction commands are generated
 - **\$sFileClass = ,./clsIpas.tpl'**; - file with the clsIpas library pattern
 - **\$sNameTestClass = ,clsTest'**; - name of a library with easy instruction command parameters test
 - **\$sNameCommClass = ,clsIpas'**; - Name of the clsIpas library
 - **\$sPostFixClass = ,class'**; - suffix for files with library (clsIpas and clsTest)
 - **\$sPostFixRequest = ,class'**; - suffix for files with instruction commands
 - **\$sRequestFcePrefix = ,IpasReq'**; - prefix for name of single instruction command functions |
\$sRequestDir = ,request'; - location of the file with instruction commands towards the library
 - **\$sDataDir = ,data'**; - location of generated files towards the generating script | **\$bGenerateRequest = true**; - switch determining if the script should generate files with instruction commands
 - **\$bGenerateCommClass = true**; - switch determining if the script should generate the clsIpas library
 - **\$sIpasServer = ,offline.ignum.cz'**; - IPAS server address
 - **\$sIpasPort = ,5055'**; - IPAS server port
 - **\$sWebCharset = ,WINDOWS-1250'**; - code page of the user interface
- With regard to the fact that the translation is done via iconv function, it is necessary to choose the code page names from the possible configurations of the function. **We caution about the fact, that with regard to the translation way it is currently not possible to choose the code page UTF-8. We will correct this defect in one of further versions of the clsIpas pattern.**

The \$sConvertChar string contains characters that the clsIpas library is going to translate into UTF-8. This string is registered in WINDOWS-1250 code page and eventual change of this code page requires a change in generating script.

Principle of library and instruction command creation

For your easier orientation it is good to know some information about the principle of library and instruction commands generating.

While the library actually won't generate and only some of the values are filled (you can recognize them easily by separators <% and %>), the files with instruction commands are completely generated. Moreover, with regard to the fact, that it comes to changeover of some instruction commands or more likely group elements into more commands, it is necessary to monitor these structures. The programme also goes through every rule, detects it's type, validation values and according to that creates a PHP code of the given function. Moreover, if it likes to use other functions that are not located in the given file, it creates a one-way connection between both files. By calling the given instruction command the file with the command is required whereby a function is defined (which consists of a prefix defined in generate.var and the instruction command name). This principle than simplifies and accelerates mainly by calling only a few instruction commands or by calling many identical commands.

The results of our tests are, that partition in more files means a more than a double saving by common connection to the server in comparison to location of all instruction commands in one file (or library) . It is due to an expressive saving of the code quantity, which has to be compiled by the system.

Together with creating these files there runs a creation of a file defaultly called clsTest.class, which contains a library that associates the most elementary validation rules defined in rule.xml.

Instruction commands

Error codes of the clsIpas library

During communication with the server via generated instruction commands it comes to validation already at the client's side, which markedly minimizes the time necessary for command processing and at the same time lowers the requests for connections to the IPAS server. The scripts always return error codes in relationship to the element which the error refers to. If it is a dive-in element, there will be an error in the multiple field too.

The main purpose always is to return maximum of error codes, to go through all validation mechanisms and eventually put together a whole code list of them. The codes are returned in a field whose key is identical with the name of the element and the values are (in case it is not the already mentioned dive-in element) located in two keys. The first and always obligate is a code key. The second, optional one is from a format keys choice, value.

Value list of code key, their meaning and the meaning of the second optional key:

- **200** Value is not a string
- **201** Value is not an integer (this control is not being used currently)
- **202** Value is not from 'True','False' values (string boolean)
- **300** Value is too short (format key contains minimal length)
- **301** Value is too long (format key contains maximal length)
- **302** Value is not from possible values (format key contains possible values separated by a comma)
- **303** Value is too small (format key contains the smallest value possible)
- **304** Value is too long (format key contains biggest value possible)
- **305** Value doesn't correspond to the prescribed data format (format key contains the prescribed data format)
- **306** Value doesn't correspond to the prescribed regular expression (format key contains the prescribed regular expression)
- **400** Empty obligate value
- **401** Unknown value not corresponding to any of possible switch values (value key contains error code of the Ipas server, which in such a case would be returned)
- **402** Data of this element have to be a field
- **403** Element in an insufficient number (format key contains minimal amount of element)
- **404** High number of element (format key contains maximal amount of element)

With regard to the number of easy validation methods will these be further implemented in newer versions and therefore more error codes will come up.

Currently error codes are divided so that they correspond to the programme position and validation type. Codes 2xx are from the easiest validation methods, code 3xx are from multiple validation methods and codes 4xx are used for errors of elements.

Instruction command list

The Ipas server currently supports following commands available for partners or sub registrars:

- **AcceptAccountAccess** – Account connection confirmation (domena.cz – payer administration)
- **AssignContact** – Import contact
- **AssignDomain** – Domain import

- **AssignSubject** – Subject import
- **AssignVirtualContact** – Virtual contact import
- **AutoRenewDomain** – Switch off/on of the domain automatic billing by expiration
- **CancelAccountAccess** – Cancelled account connection (domena.cz – payers administration)
- **CancelInvoice** – Payment notice cancellation
- **CancelOrder** – Order cancellation
- **ChangeAccountPassword** – Password change
- **CheckContact** – Detection of contact existence
- **CheckDomain** – Detection of subject existence
- **CheckSubject** – Detection of domain existence
- **ConfirmOrder** – Order confirmation
- **CreateAccount** – Account creation (user's account)
- **CreateContact** – Contact creation (will be cancelled – substituted by OrderService command)
- **CreateDomain** – Domain creation (will be cancelled – substituted by OrderService command)
- **CreateFinalInvoice** – Tax document creation
- **CreateInvoice** – Payment order creation
- **CreateNicAgreement** – Confirmation of rules for CZ domains
- **CreateNServer** – DNS server creation
- **CreateSubject** – Subject creation (will be cancelled – substituted by OrderService command)
- **CreateVirtualContact** – Virtual contact creation
- **CreditNoteGetImage** – Credit note display
- **DeleteContact** – Contact deletion (complete)
- **DeleteDomain** – Domain cancellation (complete)
- **DeleteNServer** – DNS server deletion
- **DeleteSubject** – Subject deletion (complete)
- **DeleteVirtualContact** – Virtual contact deletion
- **DetachContact** – Removing contact from the list of the n account
- **DetachDomain** – Removing domain from the list of the given account
- **DetachSubject** – Removing subject from the list of the given account
- **GetAccountInfo** – Display account information
- **GetContactInfo** – Displaycontact information according to RRID
- **GetContactInfoByID** – Display contact information according to ID
- **GetDomainInfo** – Display domain information according to domain name
- **GetDomainInfoByID** – Display domain information according to ID
- **GetNicAgreementVersions** – Return actual valid version of rules for CZ domains
- **GetSubjectInfo** – Display subject information according to RRID
- **GetSubjectInfoByID** – Display subject information according to ID
- **GetVirtualContactInfo** – Display virtual contact information
- **InvoiceGetImage** – Display payment order
- **InvoiceTaxGetImage** – Display text document
- **JoinAccount** – Integrate account
- **Login** – Log on
- **Logout** – Log out
- **MakeInvoicePayed** – Pay the payment order
- **OrderManualFinish** – Manual order confirmation
- **OrderManualStart** – Marking of the beginning of manual order processing

- **OrderService** – Service ordering
- **Query** – SELECT start-up
- **RenewDomain** – Domain prolongation (will be cancelled – substituted by OrderService command)
- **RepairRRID** – RRID correction for CZ subjects (in case it comes to occupation during order processing)
- **RequestAccountAccess** – Request for account connection (domena.cz – payer administration)
- **ResendConfirm** – Forwarding of order confirmation
- **ResendInvoice** – Forwarding of payment order/ tax document
- **SendLoginInfo** – Forwarding of log on data
- **SetCurrentAccount** – Log on under Account
- **TransferDomain** – Domain transfer (will be cancelled – substituted by OrderService command)
- **UpdateAccount** – Modify account data
- **UpdateContact** – Modify contact (will be cancelled – substituted by OrderService command)
- **UpdateDomain** – Domain modification (will be cancelled – substituted by OrderService command)
- **UpdateNServer** – DNS server modification
- **UpdateSubject** – Subject modification (will be cancelled – substituted by OrderService command)
- **UpdateVirtualContact** – Virtual contact modification
- **ViewLogin** – Display information about logged account

The rule.xml file contains also other instruction commands. However, it is not possible to call these without the Admin authorisation. By an attempt to call such an instruction command the server refuses to execute this command.

Description of important commands

Login

- **LoginName** – Log on name of the account (mandatory)
- **Password** – account password (mandatory)
- **Type** – from among Account, Partner, the System sets the account type under which you want to log on (mandatory)
- **Timeout** – sets the TimeOut connection by inactivity
- **Partner** – is a group element dedicated optionally for log on of an Account type
 - **LoginName** – log on name of the partner
 - **Password** – log on password of the partner

If an Element Partner isn't defined and if the log on type is Account, the system chooses a default partner.

GetDomainInfo

- **Domain** – name of domain whose information you want to display (mandatory)

CheckDomain

- **Domain** – is a MultiElement, which means an element that can repeat at will. It must occur at least exactly once. It contains the name of the domain about which you want to know if it is available or not. Because it is a MultiElement, the entry data in it must be a field.

Query

- **ID** – determines the SELECT that you want to start up. A list of SELECTs is available at our technical support. (mandatory).
- **Database** – there is a choice between data and log values. By data value you determine a data database, by log value you determine a log database. A default value is data.
- **Params** – is a GroupElement, which fuses all keys that are needed for a SELECT start-up. These keys are also available at our technical support, together with single SELECTs. The keys inside this element have to be identical in name with the key in SELECT. Some of the keys are automatically predefined by the system (e.g. AccID – ID of the logged user, AccLogin – contains the login of the actual user, etc.). These parameters are then overwritten by the system and there is no attention paid to the received parameters.

Examples

Demonstration of Login command via clsIpas library

```
$oMyIpas = new clsIpas();  
$aReqData[0]['name'] = ,Login`;  
$aReqData[0]['data'] = Array(,LoginName` => ,elephant`, ,Password` => ,bufamasvousama`,  
,Type` => ,Account`);  
$oMyIpas ->aRequestData = $aReqData;  
$aOutReq = $oMyIpas ->strCreateRequest();  
  
If (is_array($aOutReq)) {  
    strZpracujChyby($aOutReq); //imaginary function  
} else {  
    $oMyIpas ->Command();  
}
```

Changes

Version 2.2.1 (20051012)

Correction of the Query 'clsIpas.tpl' class

- Correction of an error in Query which ensured CodeToID translation

Version 2.2.0 (20050727)

Change of ,generate.php'

- Change of the way of generating translation fields

Version 2.1.0 (20050722)

Change of ,clsIpas.tpl'

- Correction of the error which disallowed the correct parsing of Czech characters by the XML Parser Addition of connection existence certification before the closing of this connection
- Addition of the function for recursive field translation

Version 2.0.1 (20050712)

Change of ,clsIpas.tpl'

- Correction of the error which disallowed the correct call of Exception (using @)

Version 2.0.0 (20050623)

Change of ,clsIpas.tpl'

- Modified for functionality in PHP5 (>=5.0.2)
- Addition of function support (IpasElementFunction class)
- Addition of a class with supportive functionalities (parsing XML, Query) (IpasAddFunction class)
- Possibility of SSL connection
- Error handling via Exception and IpasException

Change of ,generate.php'

- Addition of working with functions as element data
- Addition of connection type choice (generate.var)
- Transposition of ChangeLog

Version 1.1.2 (20050510)

Correction of ,generate.php'

- Exclaim error correction Inherit Element Type

Version 1.1.1 (20050415)

Correction of ,generate.php‘

- Correction of establishing of the OrderService request (translation ServiceCode and class_id)

Version 1.1.0 (20050331)

Correction of ,generate.php‘

- Change of the language conversion way
- Correction of ,quantitymax‘

Version 1.0.3 (20041026)

Correction of ,generate.php‘

- Correction of the missing ,)‘ a ,;‘ in generated request functions

Version 1.0.2 (20041025)

Correction

- Error corrected in the parameter name in the strConvertData function

Version 1.0.1 (20040924)

IPAS (20040927)

|Error corrected which avoids full-value communication in UniCode

Added

- Ability to communicate with GZIP compressed instruction commands
- Rates \$bGZData, \$iGZLevel, \$iGZMoreThan, \$aGZAllways mastering GZIPing and GZIPped output
- Rate \$sRequestCache accelerating instruction command generating (used in following functions: Command, strCreateRequest, strConvertRequest)
- trigger_error Calling in extraordinary error situations

Changes

- Instruction command returned through the strCreateRequest function is being returned in a real form. By using GZIP it is zipped (and then also contains a DWORD heading)
- Command function automatically detects type of data returned by the server and automatically unzips compressed data.
- Rename of the voidCallExternalFunction to unCallExternalFunction which corresponds to the type of returned data
- strConvertData Function newly uses strtr function method for data translation (substitutes replace str_replace function in one direction and For cyklus in the other translation direction)

Correction

- Typing error corrected in the strConvertData function (error in area preparation for data translation)