

The GNU Enterprise Application Server

Application Programmer's Interface
Edition 0.5.3, 2006-03-24

Reinhard Müller

Copyright © 2002-2006 Free Software Foundation

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Foundation.

Table of Contents

1	Introduction	1
2	Data Types	2
3	API Functions	3
3.1	Session Management	3
3.2	Handling Lists Of Objects	3
3.3	Handling Specific Objects	4
4	System Classes	5
4.1	gnue_module	5
4.2	gnue_class	5
4.3	gnue_property	5
4.4	gnue_procedure	6
4.4.1	Special procedures	6
4.4.1.1	OnInit	6
4.4.1.2	OnChange	7
4.4.1.3	OnValidate	7
4.4.1.4	onDelete	7
4.5	gnue_parameter	7
4.6	gnue_label	7
4.7	gnue_message	8
Appendix A	Data Type and Function Index	9

1 Introduction

This document describes the RPC interface provided by GNUe Appserver. Using this information, you can create a front end to GNUe Appserver.

This document is *not* about writing database applications with GNU Enterprise. You only have to read this document if you are not satisfied with GNUe Forms, and want to write a different front end to Appserver.

2 Data Types

In the API definition, we will make use of the following data type placeholders, which will have to be translated into appropriate data types for the various implemented RPC mechanisms:

void [Data Type]

This is used as the result type for functions that actually don't return a result.

boolean [Data Type]

This is a boolean data type that can only hold TRUE or FALSE values.

integer [Data Type]

This is an integer data type which must be able to contain signed 32 bit values. This restriction limits the maximum list size to more than 2 billion objects.

string [Data Type]

This is a data type that must be able to hold variable length strings without length limitations.

stringlist [Data Type]

This is an one-dimensional array of elements of the type **string**. Implementation may restrict the number of elements to 32767.

stringtable [Data Type]

This is a two-dimensional array of elements of the type **string**. Implementation may restrict the number of columns as well as the number of rows to 32767.

The RPC API is object oriented. Some of the methods described here return object instances, of which methods can be called subsequently. Handling of these objects depends on the RPC transport method.

3 API Functions

All API functions can raise exceptions on failure. All functions are atomic, in the sense that in case of an exception the function has no effect at all. No error in a function call can cause a function to be "half-done".

The exact syntax of the API functions is dependant on the selected RPC interface and the language sitting on top of it. However, we are describing the functions in a C-like syntax here, using the data type placeholders we defined above.

3.1 Session Management

session open (*auth_parameters*) [Function]

Opens a connection to Appserver using the given parameters for authentication. The number and type of the parameters still have to be decided. The return value is a session object.

An Exception is raised if the server cannot be contacted or authentication fails.

void commit () [Method on **session**]

Commits the current transaction of the session, making all changes permanent.

Exceptions can happen if a OnValidate procedure raises an exception, or if a non-nullable property is not set.

void rollback (*session_id session*) [Method on **session**]

Discards all changes done in this session since the last **commit** or **rollback**.

3.2 Handling Lists Of Objects

These functions provide a means for getting data for a list of objects fulfilling certain conditions.

list request (*string classname, stringlist conditions, stringlist sortorder, stringlist properties*) [Method on **session**]

Requests a list of objects of class *classname* matching the *conditions*. Appserver prepares to send the values of the properties listed in *properties* on subsequent calls to **fetch**, where the order of the objects is determined by the properties listed in *sort_order*. The properties in *sort_order* may, but need not appear in *properties*. *classname*, *conditions*, *sort_order*, and *properties* must contain fully referenced identifiers for classes or properties.

This function only returns a list object. No actual data is passed over the network when calling this function.

An Exception is raised if the requested class does not exist, the current user has no access to the requested class, or any of the given *properties* does not exist.

integer count () [Method on **list**]

Returns the number of objects contained in the list.

stringtable fetch (*integer start, integer count*) [Method on **list**]

Returns a 2-dimensional array of data with *count* rows, where column 0 always holds the *object_id* of the object, and the remaining columns contain the values for the properties defined in the previous call to **request**. Negative values for *start* indicate position from the end of the list. Negative values for *count* are invalid. Count may not be greater than 32767.

3.3 Handling Specific Objects

These functions provide a means for reading, writing and deleting an object or a set of objects, as well as for calling a procedure for an object or a set of objects. However, the *object_ids* for the objects to operate upon have to be determined before these functions can be used, for example by using the list handling functions described above.

stringtable load (*string classname, stringlist object_ids, stringlist properties*) [Method on **session**]

Returns a 2-dimensional array of data with a row for every entry in the *object_ids* list and a column for every entry in *properties*. Unlike **fetch**, this function does *not* automatically return the *object_ids* in column 0.

If this function is called with one of the *object_ids* being an empty string, the corresponding row in the result contains a list of the data types of the *properties*. This is a temporary hack and will be removed again in a future version!

An Exception is raised if the requested class does not exist, the current user has no access to the requested class, any of the given *object_ids* does not exist, or any of the given *properties* does not exist.

stringlist store (*string classname, stringlist object_ids, stringlist properties, stringtable values*) [Method on **session**]

Stores the data in *values* in the objects identified by *object_ids*. Every row in *values* matches an entry in *object_ids*, while every column matches an entry in *properties*. Empty object ids indicate that new objects with that data should be created. Validation is performed before the actual storing is done. If validation of a single object fails, none of the objects are stored, but an exception is raised. This function returns a list of all object ids of the stored objects. This is important for the caller to know under which object ids the new objects have been stored and can be accessed from now on. Note that after calling **store**, **commit** has to be called to make the changes persistent, while a call to **rollback** can undo the changes.

An Exception is raised if the requested class does not exist, the current user has no access to the requested class, any of the given *object_ids* does not exist, any of the given *properties* does not exist, or any of the *values* does not fit the corresponding property.

void delete (*string classname, stringlist object_ids*) [Method on **session**]

Deletes the objects of class *classname* identified through *object_ids*.

An Exception is raised if the requested class does not exist, the current user has no access to the requested class, or any of the given *object_ids* does not exist.

stringlist call (*session_id session, string classname, stringlist object_ids, string procedurename, stringlist parameters*) [Method on **session**]

Calls the procedure *procedurename* for every object identified through the *object_ids* and passes the same *parameters* to every call. The number of entries in *parameters* must match the parameter count of the procedure. This function returns a list of strings that contains the results of the procedure calls for each object.

An Exception is raised if the requested class does not exist, the current user has no access to the requested class, any of the given *object_ids* does not exist, or the given procedure is not defined.

4 System Classes

The following classes are always defined and can be accessed to query and/or influence class definitions and (in future) other information about the state of the Application Server.

4.1 `gnue_module`

- `gnue_name` *string(35)* [Property of `gnue_module`]
 The name of the module.
- `gnue_comment` *string(70)* [Property of `gnue_module`]
 Arbitrary text explaining the purpose of the module.

4.2 `gnue_class`

- `gnue_module` *gnue_module* [Property of `gnue_class`]
 The module that originally defines the class.
- `gnue_name` *string(35)* [Property of `gnue_class`]
 The name of the class without the module name. You can find out the module name by referencing the `gnue_module` property.
- `gnue_comment` *string(70)* [Property of `gnue_class`]
 Arbitrary text explaining the purpose of the class.

4.3 `gnue_property`

- `gnue_class` *gnue_class* [Property of `gnue_property`]
 The class the property belongs to.
- `gnue_module` *gnue_module* [Property of `gnue_property`]
 The module that defines the property.
- `gnue_name` *string(35)* [Property of `gnue_property`]
 The name of the property without the module name. You can find out the module name by referencing the `gnue_module` property.
- `gnue_type` *string(35)* [Property of `gnue_property`]
 The type of the property. This can be one of the predefined types "string", "number", "boolean", "date", "time", or "datetime", or the name of a class, in which case the property is a reference property to that class.
- `gnue_length` *number(6)* [Property of `gnue_property`]
 The length of the property. Only relevant if `gnue_type` is "string" or "number".
- `gnue_scale` *number(4)* [Property of `gnue_property`]
 Only relevant if `gnue_type` is "number", in which case it defines the number of fractional digits, while `gnue_length` defines the total number of digits.
- `gnue_nullable` *boolean* [Property of `gnue_property`]
 If TRUE this property can contain NULL values.
- `gnue_comment` *string(70)* [Property of `gnue_property`]
 Arbitrary text explaining the purpose of the property.

4.4 gnue_procedure

gnue_class *gnue_class* [Property of **gnue_procedure**]
The class the procedure belongs to.

gnue_module *gnue_module* [Property of **gnue_procedure**]
The module that defines the procedure.

gnue_name *string(35)* [Property of **gnue_procedure**]
The name of the procedure without the module name. You can find out the module name by referencing the **gnue_module** property.

gnue_language *string(10)* [Property of **gnue_procedure**]
Language of the procedure. Currently only "python" is valid.

gnue_code *string* [Property of **gnue_procedure**]
The source code of the procedure.

gnue_type *string(35)* [Property of **gnue_procedure**]
The type of the procedures' result. This can be one of the predefined types "string", "number", "boolean", "date", "time", or "datetime". If the procedure has no result this property will be left empty.

gnue_length *number(6)* [Property of **gnue_procedure**]
The length of the procedures' result. Only relevant if **gnue_type** is "string" or "number".

gnue_scale *number(4)* [Property of **gnue_procedure**]
Only relevant if **gnue_type** is "number", in which case it defines the number of fractional digits, while **gnue_length** defines the total number of digits.

gnue_nullable *boolean* [Property of **gnue_procedure**]
If TRUE this procedure can return "NULL" or "None" values.

gnue_comment *string(70)* [Property of **gnue_procedure**]
Arbitrary text explaining the purpose of the procedure.

4.4.1 Special procedures

Procedures having a **gnue_name** starting with "get" and without any parameters but a result are automatically treated as "calculated properties". A procedure called "getfoobar" in a module named "address" could be accessed as a property called "address_foobar". Appserver will implicitly call the procedure "address_getfoobar".

Besides this the application server looks for other special procedure names to be called automatically as "triggers". Note that one class can have several of these, all named identically, as each module can define its own trigger; in this case the base class' trigger is called first, followed by the triggers defined in other modules. The following procedure names (**gnue_name**) are recognized:

4.4.1.1 OnInit

When the application server comes to create a new instance of a given class, all procedures of this class having a **gnue_name** of "OnInit" (case-insensitive) are called. Such a procedure can refer to the instance via *self*.

4.4.1.2 OnChange

Before the application server changes the value of a property, all procedures of the class having a `gnue_name` of "OnChange" (case-insensitive) are called. The variable `propertyName` holds the name of the property to be changed. The variable `newValue` holds the new value which should be set and `oldValue` holds the original value of the property. One can use the function `abort` to prevent a modification of the property.

4.4.1.3 OnValidate

Before the application server performs a `commit` it calls all procedures of the class having a `gnue_name` of "OnValidate" (case-insensitive). Such a procedure can refer to the calling instance via `self`. One can use the function `abort` to abort the validation process and to stop the commit.

4.4.1.4 OnDelete

Before the application server deletes an instance of a class all procedures of the class having a `gnue_name` of "onDelete" (case-insensitive) will be executed. Such a procedure can refer to the calling instance via `self`. One can use the function `abort` to abort the deletion.

4.5 gnue_parameter

`gnue_procedure` *gnue_procedure* [Property of `gnue_parameter`]
The procedure the parameter belongs to.

`gnue_name` *string(35)* [Property of `gnue_parameter`]
The name of the parameter. The parameter with the name "result" defines the return value of the procedure.

`gnue_type` *string(35)* [Property of `gnue_parameter`]
The type of the parameter. This can be one of the predefined types "string", "number", "boolean", "date", "time", or "datetime". Object references cannot be passed as parameters.

`gnue_length` *number(6)* [Property of `gnue_parameter`]
The length of the property. Only relevant if `gnue_type` is "string" or "number".

`gnue_scale` *number(4)* [Property of `gnue_parameter`]
Only relevant if `gnue_type` is "number", in which case it defines the number of fractional digits, while `gnue_length` defines the total number of digits.

`gnue_comment` *string(70)* [Property of `gnue_parameter`]
Arbitrary text explaining the purpose of the parameter.

4.6 gnue_label

`gnue_property` *gnue_property* [Property of `gnue_label`]
The name attribute of the property this label is attached to.

`gnue_procedure` *gnue_procedure* [Property of `gnue_label`]
The name attribute of the procedure this label is attached to.

`gnue_language` *string(5)* [Property of `gnue_label`]
The language in which the labels are defined.

`gnue_page` *string(35)* [Property of `gnue_label`]

`gnue_label` *string(35)* [Property of `gnue_label`]

The label's text to display in the given language.

`gnue_position` *number(6)* [Property of `gnue_label`]

Designates the Y order of the label and the property/procedure it refers to.

`gnue_search` *number(6)* [Property of `gnue_label`]

If present, it makes the property referred a dropdown search field in a referring form.

`gnue_info` *number(6)* [Property of `gnue_label`]

If present, it makes the property referred a label info field in pair with the search field in a referring form.

4.7 `gnue_message`

`gnue_module` *gnue_module* [Property of `gnue_message`]

The module that defines the message.

`gnue_language` *string(5)* [Property of `gnue_message`]

The language in which the messages are defined.

`gnue_name` *string(35)* [Property of `gnue_message`]

`gnue_text` *string* [Property of `gnue_message`]

Appendix A Data Type and Function Index

B

boolean..... 2

C

call on session..... 4

commit on session..... 3

count on list..... 3

D

delete on session..... 4

F

fetch on list..... 3

I

integer..... 2

L

load on session..... 4

O

open..... 3

R

request on session..... 3

rollback on session..... 3

S

store on session..... 4

string..... 2

stringlist..... 2

stringtable..... 2

V

void..... 2