5 Loadable modules

1 Overview

Loadable modules offer a performance-minded option for extending Zabbix functionality.

There already are ways of extending Zabbix functionality by way of:

- user parameters (agent metrics)
- external checks (agent-less monitoring)
- `system.run[]` Zabbix agent item.

They work very well, but have one major drawback, namely fork(). Zabbix has to fork a new process every time it handles a user metric, which is not good for performance. It is not a big deal normally, however it could be a serious issue when monitoring embedded systems, having a large number of monitored parameters or heavy scripts with complex logic or long startup time.

Support of loadable modules offers ways for extending Zabbix agent, server and proxy without sacrificing performance.

A loadable module is basically a shared library used by Zabbix daemon and loaded on startup. The library should contain certain functions, so that a Zabbix process may detect that the file is indeed a module it can load and work with.

Loadable modules have a number of benefits. Great performance and ability to implement any logic are very important, but perhaps the most important advantage is the ability to develop, use and share Zabbix modules. It contributes to trouble-free maintenance and helps to deliver new functionality easier and independently of the Zabbix core code base.

Module licensing and distribution in binary form is governed by the GPL license (modules are linking with Zabbix in runtime and are using Zabbix headers; currently the whole Zabbix code is licensed under GPL license). Binary compatibility is not guaranteed by Zabbix.

Module API stability is guaranteed during one Zabbix LTS (Long Term Support) release cycle. Stability of Zabbix API is not guaranteed (technically it is possible to call Zabbix internal functions from a module, but there is no guarantee that such modules will work).

2 Module API

In order for a shared library to be treated as a Zabbix module, it should implement and export several functions. There are currently six functions in the Zabbix module API, only one of which is mandatory and the other five are optional.

2.1 Mandatory interface

The only mandatory function is **zbx_module_api_version()**:

```
int    zbx_module_api_version(void);
```

This function should return the API version implemented by this module and in order for the module to be loaded this version must match module API version supported by Zabbix. Version of module API supported by Zabbix is ZBX_MODULE_API_VERSION. So this function should return this constant. Old constant ZBX_MODULE_API_VERSION_ONE used for this purpose is now defined to equal ZBX_MODULE_API_VERSION to preserve source compatibility, but it's usage is not recommended.

2.2 Optional interface

The optional functions are **zbx_module_init()**, **zbx_module_item_list()**, **zbx_module_item_timeout()**, **zbx_module_history_write_cbs()** and **zbx_module_uninit()**:

```
int     zbx_module_init(void);
```

This function should perform the necessary initialization for the module (if any). If successful, it should return ZBX_MODULE_OK. Otherwise, it should return ZBX_MODULE_FAIL. In the latter case Zabbix will not start.

```
ZBX_METRIC  *zbx_module_item_list(void);
```

This function should return a list of items supported by the module. Each item is defined in a ZBX_METRIC structure, see the section below for details. The list is terminated by a ZBX_METRIC structure with “key” field of NULL.

```
void    zbx_module_item_timeout(int timeout);
```

If module exports **zbx_module_item_list()** then this function is used by Zabbix to specify the timeout settings in Zabbix configuration file that the item checks implemented by the module should obey. Here, the “timeout” parameter is in seconds.

```
ZBX_HISTORY_WRITE_CBS   zbx_module_history_write_cbs(void);
```

This function should return callback functions Zabbix server will use to export history of different data types. Callback functions are provided as fields of ZBX_HISTORY_WRITE_CBS structure, fields can be NULL if module is not interested in the history of certain type.

```
int     zbx_module_uninit(void);
```

This function should perform the necessary uninitialization (if any) like freeing allocated resources, closing file descriptors, etc.

All functions are called once on Zabbix startup when the module is loaded, with the exception of zbx_module_uninit(), which is called once on Zabbix shutdown when the module is unloaded.

2.3 Defining items

Each item is defined in a ZBX_METRIC structure:

```
typedef struct
{
    char        *key;
    unsigned    flags;
    int         (*function)();
    char        *test_param;
}
ZBX_METRIC;
```

Here, **key** is the item key (e.g., “dummy.random”), **flags** is either CF_HAVEPARAMS or 0 (depending on whether the item accepts parameters or not), **function** is a C function that implements the item (e.g., “zbx_module_dummy_random”), and **test_param** is the parameter list to be used when Zabbix agent is started with the “-p” flag (e.g., “1,1000”, can be NULL). An example definition may look like this:

```
static ZBX_METRIC keys[] =
{
    { "dummy.random", CF_HAVEPARAMS, zbx_module_dummy_random, "1,1000" },
    { NULL }
}
```

Each function that implements an item should accept two pointer parameters, the first one of type AGENT_REQUEST and the second one of type AGENT_RESULT:

```
int    zbx_module_dummy_random(AGENT_REQUEST *request, AGENT_RESULT *result)
{
    ...

    SET_UI64_RESULT(result, from + rand() % (to - from + 1));

    return SYSINFO_RET_OK;
}
```

These functions should return SYSINFO_RET_OK, if the item value was successfully obtained. Otherwise, they should return SYSINFO_RET_FAIL. See example “dummy” module below for details on how to obtain information from AGENT_REQUEST and how to set information in AGENT_RESULT.

2.4 Providing history export callbacks

History export via module is no longer supported by Zabbix proxy since Zabbix 4.0.0.

Module can specify functions to export history data by type: Numeric (float), Numeric (unsigned), Character, Text and Log:

```
typedef struct
{
    void    (*history_float_cb)(const ZBX_HISTORY_FLOAT *history, int
```

```
history_num);
    void    (*history_integer_cb)(const ZBX_HISTORY_INTEGER *history, int
history_num);
    void    (*history_string_cb)(const ZBX_HISTORY_STRING *history, int
history_num);
    void    (*history_text_cb)(const ZBX_HISTORY_TEXT *history, int
history_num);
    void    (*history_log_cb)(const ZBX_HISTORY_LOG *history, int
history_num);
}
ZBX_HISTORY_WRITE_CBS;
```

Each of them should take “history” array of “history_num” elements as arguments. Depending on history data type to be exported, “history” is an array of the following structures, respectively:

```
typedef struct
{
    zbx_uint64_t    itemid;
    int        clock;
    int        ns;
    double        value;
}
ZBX_HISTORY_FLOAT;

typedef struct
{
    zbx_uint64_t    itemid;
    int        clock;
    int        ns;
    zbx_uint64_t    value;
}
ZBX_HISTORY_INTEGER;

typedef struct
{
    zbx_uint64_t    itemid;
    int        clock;
    int        ns;
    const char    *value;
}
ZBX_HISTORY_STRING;

typedef struct
{
    zbx_uint64_t    itemid;
    int        clock;
    int        ns;
    const char    *value;
}
ZBX_HISTORY_TEXT;
```

```
typedef struct
{
    zbx_uint64_t    itemid;
    int        clock;
    int        ns;
    const char    *value;
    const char    *source;
    int        timestamp;
    int        logeventid;
    int        severity;
}
ZBX_HISTORY_LOG;
```

Callbacks will be used by Zabbix server history syncer processes in the end of history sync procedure after data is written into Zabbix database and saved in value cache.

2.5 Building modules

Modules are currently meant to be built inside Zabbix source tree, because the module API depends on some data structures that are defined in Zabbix headers.

The most important header for loadable modules is **include/module.h**, which defines these data structures. Another useful header is **include/sysinc.h**, which performs the inclusion of the necessary system headers, which itself helps include/module.h to work properly.

In order for include/module.h and include/sysinc.h to be included, the **./configure** command (without arguments) should first be run in the root of Zabbix source tree. This will create **include/config.h** file, which include/sysinc.h relies upon. (If you obtained Zabbix source code as a Subversion repository checkout, the ./configure script does not exist yet and the **./bootstrap.sh** command should first be run to generate it.)

With this information in mind, everything is ready for the module to be built. The module should include **sysinc.h** and **module.h**, and the build script should make sure that these two files are in the include path. See example “dummy” module below for details.

Another useful header is **include/log.h**, which defines **zabbix_log()** function, which can be used for logging and debugging purposes.

3 Configuration parameters

Zabbix agent, server and proxy support two parameters to deal with modules:

- LoadModulePath – full path to the location of loadable modules
- LoadModule – module(s) to load at startup. The modules must be located in a directory specified by LoadModulePath. It is allowed to include multiple LoadModule parameters.

For example, to extend Zabbix agent we could add the following parameters:

```
LoadModulePath=/usr/local/lib/zabbix/agent/
LoadModule=mariadb.so
LoadModule=apache.so
LoadModule=kernel.so
LoadModule=dummy.so
```

Upon agent startup it will load the mariadb.so, apache.so, kernel.so and dummy.so modules from the /usr/local/lib/zabbix/agent directory. It will fail if a module is missing, in case of bad permissions or if a shared library is not a Zabbix module.

4 Frontend configuration

Loadable modules are supported by Zabbix agent, server and proxy. Therefore, item type in Zabbix frontend depends on where the module is loaded. If the module is loaded into the agent, then the item type should be “Zabbix agent” or “Zabbix agent (active)”. If the module is loaded into server or proxy, then the item type should be “Simple check”.

History export through Zabbix modules does not need any frontend configuration. If the module is successfully loaded by server and provides **zbx_module_history_write_cbs()** function which returns at least one non-NULL callback function then history export will be enabled automatically.

5 Dummy module

Zabbix includes a sample module written in C language. The module is located under src/modules/dummy:

```
alex@alex:~trunk/src/modules/dummy$ ls -l
-rw-rw-r-- 1 alex alex 9019 Apr 24 17:54 dummy.c
-rw-rw-r-- 1 alex alex   67 Apr 24 17:54 Makefile
-rw-rw-r-- 1 alex alex  245 Apr 24 17:54 README
```

The module is well documented, it can be used as a template for your own modules.

After ./configure has been run in the root of Zabbix source tree as described above, just run **make** in order to build **dummy.so**.

```
/*
** Zabbix
** Copyright (C) 2001-2016 Zabbix SIA
**
** This program is free software; you can redistribute it and/or modify
** it under the terms of the GNU General Public License as published by
** the Free Software Foundation; either version 2 of the License, or
** (at your option) any later version.
**
** This program is distributed in the hope that it will be useful,
** but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
** GNU General Public License for more details.
**
** You should have received a copy of the GNU General Public License
** along with this program; if not, write to the Free Software
** Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-
1301, USA.
**/

#include "sysinc.h"
#include "module.h"

/* the variable keeps timeout setting for item processing */
static int    item_timeout = 0;

/* module SHOULD define internal functions as static and use a naming
pattern different from Zabbix internal */
/* symbols (zbx_*) and loadable module API functions (zbx_module_*) to avoid
conflicts                    */
static int    dummy_ping(AGENT_REQUEST *request, AGENT_RESULT *result);
static int    dummy_echo(AGENT_REQUEST *request, AGENT_RESULT *result);
static int    dummy_random(AGENT_REQUEST *request, AGENT_RESULT *result);

static ZBX_METRIC keys[] =
/*    KEY            FLAG        FUNCTION    TEST PARAMETERS */
{
    {"dummy.ping",        0,        dummy_ping,    NULL},
    {"dummy.echo",        CF_HAVEPARAMS,    dummy_echo,    "a message"},
    {"dummy.random",    CF_HAVEPARAMS,    dummy_random,    "1,1000"},
    {NULL}
};

/******************************************************************************
***
 *
*
 * Function: zbx_module_api_version
*
 *
*
 * Purpose: returns version number of the module interface
*
 *
*
 * Return value: ZBX_MODULE_API_VERSION - version of module.h module is
*
 *               compiled with, in order to load module successfully Zabbix
*
 *               MUST be compiled with the same version of this header file
*
 *
```

```
*
******************************************************************************
**/
int    zbx_module_api_version(void)
{
    return ZBX_MODULE_API_VERSION;
}

/******************************************************************************
***
 *
*
 * Function: zbx_module_item_timeout
*
 *
*
 * Purpose: set timeout value for processing of items
*
 *
*
 * Parameters: timeout - timeout in seconds, 0 - no timeout set
*
 *
*
******************************************************************************
**/
void   zbx_module_item_timeout(int timeout)
{
    item_timeout = timeout;
}

/******************************************************************************
***
 *
*
 * Function: zbx_module_item_list
*
 *
*
 * Purpose: returns list of item keys supported by the module
*
 *
*
 * Return value: list of item keys
*
 *
*
```

```
******************************************************************************
**/
ZBX_METRIC  *zbx_module_item_list(void)
{
    return keys;
}

static int    dummy_ping(AGENT_REQUEST *request, AGENT_RESULT *result)
{
    SET_UI64_RESULT(result, 1);

    return SYSINFO_RET_OK;
}

static int    dummy_echo(AGENT_REQUEST *request, AGENT_RESULT *result)
{
    char    *param;

    if (1 != request->nparam)
    {
        /* set optional error message */
        SET_MSG_RESULT(result, strdup("Invalid number of parameters."));
        return SYSINFO_RET_FAIL;
    }

    param = get_rparam(request, 0);

    SET_STR_RESULT(result, strdup(param));

    return SYSINFO_RET_OK;
}

/******************************************************************************
***
 *
*
 * Function: dummy_random
*
 *
*
 * Purpose: a main entry point for processing of an item
*
 *
*
 * Parameters: request - structure that contains item key and parameters
*
 *              request->key - item key without parameters
*
 *              request->nparam - number of parameters
*
 *              request->timeout - processing should not take longer than
```

```
 *                                     this number of seconds
 *
 *              request->params[N-1] - pointers to item key parameters
 *
 *
 *
 *             result - structure that will contain result
 *
 *
 *
 * Return value: SYSINFO_RET_FAIL - function failed, item will be marked
 *                                     as not supported by zabbix
 *
 *               SYSINFO_RET_OK - success
 *
 *
 *
 * Comment: get_rparam(request, N-1) can be used to get a pointer to the Nth
 *          parameter starting from 0 (first parameter). Make sure it exists
 *          by checking value of request->nparam.
 *
 *
 *
 ******************************************************************************
**/
static int    dummy_random(AGENT_REQUEST *request, AGENT_RESULT *result)
{
    char    *param1, *param2;
    int    from, to;

    if (2 != request->nparam)
    {
        /* set optional error message */
        SET_MSG_RESULT(result, strdup("Invalid number of parameters."));
        return SYSINFO_RET_FAIL;
    }

    param1 = get_rparam(request, 0);
    param2 = get_rparam(request, 1);

    /* there is no strict validation of parameters for simplicity sake */
    from = atoi(param1);
    to = atoi(param2);

    if (from > to)
```

```
    {
        SET_MSG_RESULT(result, strdup("Invalid range specified."));
        return SYSINFO_RET_FAIL;
    }

    SET_UI64_RESULT(result, from + rand() % (to - from + 1));

    return SYSINFO_RET_OK;
}

/******************************************************************************
***
 *
*
 * Function: zbx_module_init
*
 *
*
 * Purpose: the function is called on agent startup
*
 *          It should be used to call any initialization routines
*
 *
*
 * Return value: ZBX_MODULE_OK - success
*
 *               ZBX_MODULE_FAIL - module initialization failed
*
 *
*
 * Comment: the module won't be loaded in case of ZBX_MODULE_FAIL
*
 *
*
*

******************************************************************************
**/
int    zbx_module_init(void)
{
    /* initialization for dummy.random */
    srand(time(NULL));

    return ZBX_MODULE_OK;
}

/******************************************************************************
***
 *
*
 * Function: zbx_module_uninit
*
```

```
 *
*
 * Purpose: the function is called on agent shutdown
*
 *          It should be used to cleanup used resources if there are any
*
 *
*
 * Return value: ZBX_MODULE_OK - success
*
 *               ZBX_MODULE_FAIL - function failed
*
 *
*

******************************************************************************
**/
int    zbx_module_uninit(void)
{
    return ZBX_MODULE_OK;
}

/******************************************************************************
***
 *
*
 * Functions: dummy_history_float_cb
*
 *            dummy_history_integer_cb
*
 *            dummy_history_string_cb
*
 *            dummy_history_text_cb
*
 *            dummy_history_log_cb
*
 *
*
 * Purpose: callback functions for storing historical data of types float,
*
 *          integer, string, text and log respectively in external storage
*
 *
*
 * Parameters: history     - array of historical data
*
 *             history_num - number of elements in history array
*
 *
*
```

```
******************************************************************************
**/
static void   dummy_history_float_cb(const ZBX_HISTORY_FLOAT *history, int
history_num)
{
    int    i;

    for (i = 0; i < history_num; i++)
    {
        /* do something with history[i].itemid, history[i].clock,
history[i].ns, history[i].value, ... */
    }
}

static void   dummy_history_integer_cb(const ZBX_HISTORY_INTEGER *history,
int history_num)
{
    int    i;

    for (i = 0; i < history_num; i++)
    {
        /* do something with history[i].itemid, history[i].clock,
history[i].ns, history[i].value, ... */
    }
}

static void   dummy_history_string_cb(const ZBX_HISTORY_STRING *history, int
history_num)
{
    int    i;

    for (i = 0; i < history_num; i++)
    {
        /* do something with history[i].itemid, history[i].clock,
history[i].ns, history[i].value, ... */
    }
}

static void   dummy_history_text_cb(const ZBX_HISTORY_TEXT *history, int
history_num)
{
    int    i;

    for (i = 0; i < history_num; i++)
    {
        /* do something with history[i].itemid, history[i].clock,
history[i].ns, history[i].value, ... */
    }
}
```

```
static void	dummy_history_log_cb(const ZBX_HISTORY_LOG *history, int history_num)
{
	int	i;

	for (i = 0; i < history_num; i++)
	{
		/* do something with history[i].itemid, history[i].clock, history[i].ns, history[i].value, ... */
	}
}

/******************************************************************************
 *                                                                            *
 * Function: zbx_module_history_write_cbs                                     *
 *                                                                            *
 * Purpose: returns a set of module functions Zabbix will call to export     *
 *          different types of historical data                                *
 *                                                                            *
 * Return value: structure with callback function pointers (can be NULL if   *
 *               module is not interested in data of certain types)           *
 *                                                                            *
 ******************************************************************************/
ZBX_HISTORY_WRITE_CBS	zbx_module_history_write_cbs(void)
{
	static ZBX_HISTORY_WRITE_CBS	dummy_callbacks =
	{
		dummy_history_float_cb,
		dummy_history_integer_cb,
		dummy_history_string_cb,
		dummy_history_text_cb,
		dummy_history_log_cb,
	};

	return dummy_callbacks;
}
```

The module exports three new items:

- `dummy.ping` - always returns '1'
- `dummy.echo[param1]` - returns the first parameter as it is, for example, `dummy.echo[ABC]` will return ABC
- `dummy.random[param1, param2]` - returns a random number within the range of param1-param2, for example, `dummy.random[1,1000000]`

6 Limitations

Support of loadable modules is implemented for the Unix platform only. It means that it does not work for Windows agents.

In some cases a module may need to read module-related configuration parameters from *zabbix_agentd.conf*. It is not supported currently. If you need your module to use some configuration parameters you should probably implement parsing of a module-specific configuration file.