AutoYaST Configuration Management System (CMS)

Anas Nashif

AutoYaST2 CMS now has a new interface which enables adding modules in execution time, depending on what modules are installed on the System. To enable this functionality, the various YaST2 run-time modules have to be adapted. CMS code that dealt with the different modules has moved to the run-time modules to keep such parts of the code in sync with the development being done on these modules.

1. Introduction

The new plugin-like design has the following advantages:

- New Run-time module can be integrated automatically in the CMS, thus new features in YaST2 mean new features in the auto-installer.
- Only installed run-time modules are being offered in the CMS. This allows the integration of speciality run-time modules which are only present on business products for example.
- Code which is relevant to one run-time configuration module is kept in one place instead of maintaining it in different modules.

2. The run-time module interface in CMS

The new interface (AKA _auto.ycp client) is now similar to the proposal structure and consists of functions. All the dialog code has been moved to autoyast to simplify the interface and to provide a common user interface for all modules appearing in the autoyast configuration system.

The interface of a run-time configuration module in the CMS has the follwoing components:

- *Summary Area,*: Contains a summary of the consifguration with the values if available. If values where not configured, the phrase *'Not configured yet'* is used, which is available from the summary module.
- *Configuration Button*: A button which calls the module in auto mode (<*module name>_auto.ycp*).
- *Reset Button*: A button for resetting the configuration data. This will delete only data in the running module.

The summary area is filled with data from the *Summary* function in the module controling the configuration. (i.e *NIS::Summary()* in the NIS package).

3. Run-time modules in *auto* mode

The <module name>_auto.ycp client accepts 2 argument:

- Function
- Configuration Data

The following functions are needed to make any module work in autoyast:

- *Import:*

  Import existing data into the module, usually done only once at the beginning.

- *Summary:*

  To provide a brief summary of the configuration. Calls >Module<::Summary()

- *Reset:*

  Resets the configuration. It returns empty values but it also can return default values, depending on the module.

- *Change:*

  This function starts the widget sequence

- *Write*

  Write the configuration without displaying any widgets and popups and without restarting any services etc. Calls <Module>::Write (and sets <Module>::write_only true)

- *Export:*

  Returns the current configuration Calls <Module>::Export

The following example shows <module name>_auto.ycp with the changes needed for the new behaviour.

```
/**
 * @param function to execute
 * @param map/list of XXpkgXX settings
 * @return map edited settings, Summary or boolean on success depending on called function
 * @example map mm = $[ "FAIL_DELAY" : "77" ];
 * @example map ret = WFM::CallFunction ("XXpkgXX_auto", [ "Summary", mm ]);
 */

{

textdomain "XXpkgXX";

y2milestone("----------------------------------------");
y2milestone("XXPkgXX auto started");

import "XXPkgXX";
include "XXpkgXX/wizards.ycp";

any ret = nil;
string func = "";
map param = $[];

/* Check arguments */
if(size(Args()) > 0 && is(Args(0), string)) {
    func = WFM::Args(0);
    if(size(Args()) > 1 && is(Args(1), map))
 param = WFM::Args(1);
}
y2debug("func=%1", func);
y2debug("param=%1", param);

/* Create a  summary*/
if(func == "Summary") {
    ret = select(XXPkgXX::Summary(), 0, "");
}
else if (func == "Import") {
    ret = XXPkgXX::Import($[]);
    return ret ;
}

/* Reset configuration */
else if (func == "Reset") {
    XXPkgXX::Import($[]);
    ret = $[];
}
/* Change configuration (run AutoSequence) */
else if (func == "Change") {
    ret = XXPkgXXAutoSequence();
}
/* Return actual state */
else if (func == "Export") {
    ret = XXPkgXX::Export();
}
/* Write givven settings */
else if (func == "Write") {
    import "Progress";
    XXPkgXX::write_only = true;
    Progress::off();
    ret = XXPkgXX::Write();
    Progress::on();
    return ret;
}
/* Unknown function */
else {
    y2error("Unknown function: %1", func);
    ret = false;
}

y2debug("ret=%1", ret);
y2milestone("XXPkgXX auto finished");
y2milestone("----------------------------------------");

return ret;

/* EOF */
}
```

Example 1. XXpkgXX_auto.ycp (XXpkgXX = module name)

4. Module functions needed for Auto-Install

The follwoing is the list of function that should be available with the modules to provide the functionality needed in the CMS.

Import()

Name

`Import()` — Imports settings from arguments

Description

This function imports settings from arguments and sets the module variables. The *Import* function should bring the module to a state where the data imported is enough for the module to write (See Write()) a usable configuration.

Returns

Import() return false if some of the required keys are missing in the imported map. If the imported data is empty, it returns true and starts the module using default values.

Export()

Name

`Export()` — Exports configured data to calling module

Description

Exports configured data to calling module. This function also converts the internal variables to unique and human readable variables which can be used in the control file needed for the auto-installation. It is not allowed to export *maps* with configured data as the keys. Configured data must be the value of a variable.

Example

```
   global define map Export () “{
return $[
    "start_nis": start,
    "nis_servers": servers,
    "nis_domain": domain,
    "start_autofs": _start_autofs,
    ];
   }
```

Example 1. Export()

Returns

This function returns a map of the configuration data.

Write()

Name

`Write()` — Write or commit configured or imported data

Description

Commit configured data. This function is also used in normal operation mode and should not be used to write configuration in auto-installation mode. Instead, use *WriteOnly()* or set the global variable *write_only* to true before writing.

WriteOnly()

Name

`WriteOnly()` — Only write/commit data, do not restart any services yet

Write configuration only without restarting services or make the system use the new configuration. This is important in the so called *continue mode*, as services are started directly after the YaST2 has finished installation anyways.

5. Configuration file

When the CMS is invloked, it checks for the configuration files in `/usr/share/YaST2/config` and evaluates them to later include them in the configuration interface.

AutoYaST2 CMs uses the same configuration file used for the YaST2 Control Center with some additions and modifications.

The following is an example of the configuration file for the NIS module:

```
;
; Menuentry file used by
;  YaST2 Control Center
;
; $Id: design.xml,v 1.5 2003/02/11 04:05:57 nashif Exp $
;

[Y2Module nis]

; name of the entry, shown in y2cc
Name = _("NIS client")

; "raw" group name to which this module belongs
; see /usr/lib/YaST2/etc/y2controlcentericons.y2cc for valid values
Group = Net_advanced

; icon displayed in y2cc
Icon = nis_client.png

; *one* *short* descriptive line for y2cc
Helptext = _("Configure NIS client")

; arguments, seperated by space
Arguments =

; whether root privileges are required to run this module
RequiresRoot = true

; default size for window, Y2CC will set $Y2_GEOMETRY to this value
Geometry =

; string to use for sorting instead of Name
SortKey =

; which textdomain to use for translations (important!)
Textdomain = nis

; vim:syntax=dosini
```

Example 3. Auto-Install Configuration file for NIS

In addition to the keywords from the last example, the CMS also evaluates the following keywords:

Autoinst

Name

`Autoinst` — Is the module compatible with the CMS and can Import/Export configurations?

Values:

- *all*: Full auto-installation support, including configuration (_auto) and writing (_write)
- *write*: Write only support
- *configure*: Configuration only support. Normally used only with parts related to installation like partitioning and general options which have no run-time module with support

for auto-installation. Data is written using the common installation process and modules available in YaST2

AutoinstPath

Name

`AutoinstPath` — Path in the control file

Values:

configure or install: All run-time configuration modules are contained in the *configure* resource. Only configuration data directly touching the installation of a system are contained in the *install* resource.

AutoClient

Name

`AutoClient` — NAme of the client to call

Values

Name of the client to be called by the CMS.

Default Value

<module name>_auto

6. Conventions for module Writers

6.1. Exported Data

- *Type of exported data*:

  Modules should only export data which is normally selected or entered by the user in normal module operation. No computed or automatically probed data should be exported.

- *Use Namespaces*

  Exported variables should have a unique name when possible and when general terminology is being used. To avoid conflicts and confusion, use a name space identifier with common words. For example, if a module should export the variable name *options*, it is better to export *<module name>.options* to avoid confusion with other modules using *options*, which is very common in configurations.

- *Lower case variables*

  To have a common and unified look of the control file, please use lower case variables when exporting the configuration data.

- The structure of the exported data should be readable and not unnecessarly complex.
- Avoid using configuration data as the key in a map key/value pair. The key of the pair must always contain the variable name, rather than it's contents

6.2. YaST2 Module Types

YaST2 configuration modules and in relation with AutoYaST can be put into three categories:

1. Simple modules which mormally only change sysconfig variable and have simple configuration data structure. (i.e. mail, nis, ldap, etc.)

   This category needs no special attention and is easy to integrate with the AutoYaST.

2. Simple modules dealing with hardware configuration (i.e. network, sound, printer etc.)

   These modules need to be able to read and autodetect hardware data during installation if no hardware data is specified in the control file. The behaviour of this type of modules up to 8.1 was to import data and write it wihtout actually reading anything from the system.

   An additional step has to be added between the import and the write, where hardware data is read and imported into the module. In some case this is simply done by calling the Read function the module.

3. Modules for management of complex configuration files (i.e. inetd, sysconfig, runlevel, users, bootloader)

   This class of modules is much more complex and requires adaptation and special attention. AutoYaST expects that only new and modified entries will be exported and not the whole configuration tree. For example when a user enables a service in inetd, only this service is exported. A user should be able to add new services which are not available in the default configuration file too.

6.3. Module behaviour

In configuration mode for auto-installation, modules *should not*(configuration system is the machine where the control file is being created):

- Read any data from the configuration system
- Probe or detect hardware on the configuration system
- Change configuration data on the configuration system
- Offer a link to other modules (i.e. calling the NIS module from the users module)
- Check if a needed package is installed on the configuration system.