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 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 buttom for resetting the configuration data. This will delete only data for 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 1 argument which has to be a *map* (struct) with the configuration data if available or an empty map, if the data is configured for first time.

The <module name>_auto.ycp client returns a list which has the latest user input, i.e. *'next* or *'back* as the first element. The second element contains the configuration data in the form of a map.

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

```
{
    textdomain "nis";

    import "Nis";
    import "Wizard";
    include "nis/ui.ycp";
```

```
    list args = Args ();
    if ( size (args) <= 0 )
    {
y2error ("Did not get the settings, probably some mistake...");
return false;
    }
    if ( !is ( Args (0), map ) )
    {
y2error ("Bad argument for nis_auto: %1", Args (0));
return false;
    }
    map settings = $[];
    {
integer i = 0;
while (i < size (Args()))
{
    if (is (Args (i), map) && nil != Args (i)) settings = Args (i);
    i = i + 1;
}
    }
    y2milestone("Imported: (%1)", settings);
    Nis::Import ( settings );

    define set_contents()“{
term contents =
    ‘VBox(
  ‘VSpacing(1),
  ‘RichText( ‘id(‘summary), Nis::Summary()),
  ‘VSpacing(0.5),
  ‘HBox(
‘PushButton(‘id(‘configure), _("&Configure NIS")),
‘HStretch(),
‘PushButton(‘id(‘reset), _("&Reset Configuration"))
),
  ‘VSpacing(1)
  );

Wizard::SetContents(_("NIS Configuration"),
    contents, "", true, true);
    }


    Nis::_autofs_allowed = true;
    set_contents();
    any result = nil;
    any ret = nil;
    repeat {
ret = UI::UserInput();
if (ret == ‘configure) {


        Wizard::CreateDialog ();
Wizard::ReplaceAbortButton(‘Empty ());
result = NisDialog ();


UI::CloseDialog ();
if (result == ‘next || result == ‘finish)
{
     settings = Nis::Export ();
}
Nis::Set(settings);
set_contents();
} else if ( ret == ‘reset) {
    settings= $[];
    Nis::Set(settings);
    set_contents();
}

y2milestone("ret: %1", ret);
    } until (ret == ‘back || ret == ‘next || ret ==‘key || ret == ‘finish);

    return [ret, settings];
}
```

Example 1. nis_auto.ycp

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.

Example

```
    global define boolean Import (map settings) “{
if (size (settings) == 0)
{
    //Provide defaults for autoinstallation editing:
    //Leave empty.
    old_domain = domain;
    return true;
}

boolean missing = false;
foreach (‘k, ["start_nis", "nis_domain", "nis_servers","start_autofs"], “{
    if (! haskey (settings, k))
    {
y2error ("Missing at Import: ’%1’.", k);
missing = true;
    }
});
if (missing)
{
    return false;
}

Set(settings);
return true;
    }
```

Example 1. Import()

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.

Set()

Name

Set()— Set the module data in a specific state

Description

Instead of using*Import(),Set()* is needed if module has to be set to a specific state without checking the validity of the data. This is required when resetting the module configuration data. This function is used in *Import()* to set the module with the imported data after validity checking has been done.

Example

```
    global define void Set (map settings) “{
        start = lookup (settings, "start_nis", false);
        servers = lookup (settings, "nis_servers", []);
        domain = lookup (settings, "nis_domain", "");
        old_domain = domain;
        // autofs is not touched in Write if the map does not want it
        _autofs_allowed = haskey (settings, "start_autofs");
        _start_autofs = lookup (settings, "start_autofs", false);
    }
```

Example 1. Set()

Returns

void

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()*.

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/lib/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 1.4 2002/04/22 04:40:27 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 5. 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 use 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 using general terminology. 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.