Text Installer Design Document

Version 2.0.9: June 17, 2010, 01:53 PM

This document defines the design specification for the Text Installer project. The project website is located at http://opensolaris.org/os/project/caiman/TextInstallerProject/

Table of Contents

1: Overview

The OpenSolaris Text Installer project will provide an interactive text-based installation experience. This project targets users installing onto server machines, but will also be useful for any user installing onto SPARC machines, or as an alternative to the liveCD on x86. As the primary target is server installations, the Text Installer will install a basic set of packages suitable for most servers.

2: Components

The following sections describe the major components of the Text Installer. Major functions, input and output, major exception cases, and pseudo-code are present where relevant.

2.1: nCurses screens

2.1.1: List of Screens

The nCurses screens provide the direct UI of the Text Installer. The fully featured product will have the following screens:

- Welcome/Navigation
- Disk Selection
- Fdisk Partition List (x86 only)
- Fdisk Partition Management (x86 only)
- Slice List
- Slice Management
- Network Type
- ~~Static IP Configuration~~
- Time Zone - Region
- Time Zone - Country
- Time Zone Selection
- Date/Time Setting
- ~~Root Password and~~ User Account
- Summary and Pre-Installation Review
- Installation Progress
- Installation Results
- An associated Help Screen for each of the above
- Install Log Viewer

A complete description of each screen's functionality, including a representative screenshot for each, is available at: http://www.opensolaris.org/os/project/caiman/TextInstallerProject/UI_Specification/

2.1.2: UI Components

The following describes the layout and structure of the objects used to create and display the nCurses screens. Trivial classes, attributes and functions are omitted.

- Class MainWindow
  - Instance Attributes
    - theme: Defines the color theme for the main window and its children
    - screen_list: Defines the screens to display
    - default_cursor_loc: Location on screen to move the cursor if cursor cannot be hidden
    - footer: InnerWindow representing the screen footer
    - header: InnerWindow representing the screen header
    - central_area: InnerWindow representing the main portion of the screenshot
    - popup_win: InnerWindow used to display pop-up dialogs
    - error_line: InnerWindow reserving space on screen for the display of errors
    - continue_action, help_action, back_action, quit_action: Default Actions for each screen
    - actions: List of current actions displayed
  - Functions
    - reset: Resets the MainWindow to its initial state
    - reset_actions: Resets the continue_action, back_action, help_action and quit_action attributes to their defaults
    - clear: Clears the window of text and input fields
    - set_header_text: Sets the header
    - set_default_actions: Resets actions to the list of default actions
    - show_actions: Updates the screen to display Actions described in actions
    - process_input: Provides a main loop, retrieving input from user and passing it to child windows for processing, until a new screen is requested
    - pop_up: Displays the popup_win with given text and options, and returns the selected option
- Class InnerWindow
  - Class variables
    - KEY_TRANSLATE: Dictionary mapping of keystrokes

    - Instance Attributes
        - border_size: Tuple representing the border width. All positional references to this window (such as when adding text) are based off the area of this window excluding this border
        - selectable: Boolean indicating if this InnerWindow can be made an 'active' UI element
        - on_make_active, on_make_inactive: Pointers to generic functions that should be called by make_active and make_inactive
        - on_make_active_kwargs, on_make_inactive_kwargs: Dictionary of keyword arguments to pass to the on_make_(in)active functions.
        - objects: List of sub-windows that are 'selectable'
        - all_objects: List of all sub-windows, including those that are displayed but not selectable
        - active_object: Index of the currently active UI element within this window
        - key_dict: Mapping of keystrokes to function calls
        - area: A WindowArea representing the portion of the terminal window that this InnerWindow covers
        - color_theme: The ColorTheme for this window
    - Functions
        - make_active, make_inactive: Repaints this window to be an (in)active UI element
        - add_text: Adds a single line of text to the window
        - add_paragraph: Adds a paragraph of text to the window, wrapping on whitespace
        - clear: Clears this window of all text and UI elements
        - on_key_down, on_key_up: Functions to handle up/down arrow keystrokes
        - process: Processes keystrokes, passing down to children UI elements (if any)
- Class ColorTheme
    - Instance Attributes
        - default: Color used by default UI elements
        - border: Color used by the screen border
        - header: Color used by the screen header
        - edit_field: Color used by editable fields
        - highlight_edit: Color used when an editable field is the active UI element
        - list_field: Color used when an item in a list is the active UI element
        - error_msg: Color used when by error messages
        - progress_bar: Color used by progress bars

- Class EditField (subclass of InnerWindow)
  - Instance Attributes
    - right_justify: Boolean indicating if the text in this field should be right justified
    - numeric_pad: When right_justify is True, the text is padded with this value
    - validate, validate_kwargs: Generic function pointer and function arguments called to validate that the text in this field is syntactically valid
    - on_exit, on_exit_kwargs: Generic function pointer and arguments called to semantically validate this field after the user has finished typing
  - Functions
    - handle_input: Processes keystrokes and checks validity before printing to screen
    - is_valid: Checks the length and “validate” functions to determine if the current text entered is syntactically valid
- Class ListItem (subclass of InnerWindow)
  - Functions:
    - set_text: Sets the text displayed by this item in the list
- Class ScreenList
  - Instance Attributes
    - screen_list: A list of screen objects
    - visited_list: A subset of screen_list tracking which screens have already been visited
    - help: A screen object, not in the flow of screen_list, that represents the screen that should be displayed if a user selects to look at the help text.
  - Functions
    - get_next: Returns the next screen to show
    - previous_screen: Returns the last visited screen, removing it from the visited_list
    - peek_last: Returns the last visited screen without removing it from the visited_list
    - quit: Returns a value indicating that the program should exit
    - show_help: Prepares and returns the “help” screen
- Class Action
  - Instance Attributes
    - key: Keystroke corresponding to this action
    - text: Display text representing this action
    - do_action: Pointer to a function that should be executed when this Action's key is pressed. This generic function should return a screen.

In summary, there will be a generic MainWindow object. The MainWindow object captures the common needs of each screen – containing header, footer, and saving space on the screen for error messages. The header is simply a title and color combination. In the footer, the user can register a list of "Actions." An Action is a class which ties a key press to a dynamically assigned function. By default, F2_Continue, F3_Back, F6_Help and F9_Quit will be registered. Additional actions can be added on a per screen basis; certain screens may also choose to remove actions (for example, the first screen would remove F3_Back). The error_line is much like the header, but can also be hidden or shown based on the user's actions.

Each screen will take the MainWindow, add header text and make adjustments to footer actions as needed, then add InnerWindow based UI elements and/or text to the central_area.

2.1.3: Screen Navigation

Managing screen navigation is done by the ScreenList class, which tracks all screens that could possibly be shown (in order), and the screens that have been visited. The MainWindow manages the screen list, and the default set of Actions have functions that are mapped to functions of the ScreenList class – for example, the do_action function associated with MainWindow.continue_action is ScreenList.get_next, returning the next screen.

At a higher level, the "main" function of the Text Installer will have a loop that looks something like the following:

```
screen = first_screen
while screen is not None
     screen = screen.show()
```

What this demonstrates is that each screen, after performing its work, will return what the next screen should be. Note that each screen has a reference to MainWindow which is used to access the Actions and other common functionality provided by that object. Thus, screen.show() works something like the following pseudo-code:

```
def show(...):
     main_window.clear()
     <prepare_ui_elements>
     return main_window.process_input()
```

Note that the process_input method of MainWindow loops on keystrokes until a keystroke associated with an Action is pressed, at which point the Action's function is called and the resulting screen returned. (Non-action associated keystrokes are passed down to child UI elements until they are handled or ignored).

2.1.4: Help Screen Navigation

Help Screens will not be part of the navigational flow. When the user requests help, a new window is laid on top of the body of whatever screen is currently displayed. The Actions for the current window are replaced with Help Screen Actions. The window will initially show text that is relevant to the current installer screen. The user may then elect to either return to the installer, or view the Help Index. The Help Index allows the user to view Help Topic information for any of the screens of the installer. Regardless of which Help Topics the user views, exiting from Help will remove the Help window, revealing the last installer screen they had been working with. At this time, the Help "Actions" will be unregistered, and any Actions associated with the current installer screen will be re-registered.

2.1.5: Exiting the Installer – Keyboard Interrupts

Users will be required to exit the Text Installer by hitting F9. The keyboard interrupt, CTRL-C, will be disabled during execution, except on the first screen to provide a fallback for possible TERM environment issues.

2.1.6: Error Conditions

If the user inputs a syntactically invalid value or makes an invalid selection, they will be notified using the MainWindow.error_line interface described above.

If an unhandled, fatal error occurs during installation, the text installer exits with a message telling the user that an unhandled exception occurred , that full traceback information is in the install log, and suggesting a bug be filed.

Screen specific error handling (when/where errors appear, etc), is described in the UI Specification at: http://www.opensolaris.org/os/project/caiman/TextInstallerProject/UI_Specification/

2.2: Installation Profile Python Module

2.2.1: Overview

A new Python object, called an InstallationProfile, will be used to track the user's selections through the Text Installer screens.

2.2.2: InstallationProfile Structure

An InstallationProfile will be comprised of four sub-classes: DiskInfo, SystemInfo, NetworkInfo, Zpool, and UserInfo. Each sub-class is delivered as a separate class within the same Python module – thus, any sub-class can be used and referenced as an independent object. In addition to the previously mentioned sub-classes, PartitionInfo and SliceInfo classes will also be created, used within DiskInfo. These may also be accessed independently. The ZFSDataset will also be created, used within Zpool.

- DiskInfo [] disks
- SystemInfo system
- NetworkInfo network
- Zpool

- UserInfo [] users

2.2.2.1: DiskInfo

- Instance Attributes
  - name: Disk name (e.g. c0t0d0)
  - size: Total size of the disk
  - use_whole_segment: Boolean indicating if the entire disk will be used for installation
  - partitions: List of ParititionInfo objects
  - slices: List of SliceInfo objects (only one of slices or partitions will have data)
- Functions
  - get_solaris_data: Returns the slice or partition target for installation, if one is set
  - get_extended_partition: Returns the extended partition, if there is one
  - get_logicals, get_standards: Returns the subset of partitions that represents the logical or standard partitions, respectively
  - remove_logicals: Removes all logical partitions from the disk
  - collapse_unused_logicals: “Collapses” adjacent unused logical partitions, combining them
  - add_unused_parts: Adds SliceInfo or PartitionInfo objects representing the unclaimed disk space
  - sort_disk_order: Sorts the slices or partitions in disk layout order (by offset)
  - get_parts: Returns either the list of slices or partitions, depending on what this disk has
  - to_tgt: Returns a tgt.Disk representation of this DiskInfo
  - create_default_layout: Creates a “default” layout for this disk, removing any slices or partitions and replacing them a generic full disk layout.

2.2.2.1.1: PartitionInfo

- Class Variables
  - SOLARIS, SOLARIS_ONE, EXT_DOS, FDISK_EXTLBA: integers representing fdisk partition types
  - EXTENDED: List of partition types that represent extended partitions
- Instance Attributes
  - id: The id of this partition (integer representing, e.g. SOLARIS)
  - number: The number of this partition

- size: In bytes
- offset: “Distance” in bytes from the beginning of the disk where this partition is. Logical partitions also indicate offset from beginning of disk.
- use_whole_segment: Boolean indicating whether the whole partition should be used for installation, or a slice within the partition.
- Slices: List of SliceInfo objects representing the slice layout of this partition

- Functions
  - is_logical, is_extended, is_solaris_data: Return True if this partition is a logical partition, extended partition, or Solaris2 partition, respectively
  - editable: Returns True if the user is allowed to modify the size of this partition during installation.
  - add_unused_parts: Adds SliceInfo objects to the slices list representing any unused space in this partition.
  - adjust_offset: Adjusts the offset of this partition slightly, ensuring that it doesn't overlap with adjacent partitions due to rounding errors.
  - modified: Returns True if the user has modified the layout of this partition in some way
  - destroyed: Returns True if the data on this partition will be destroyed as a result of modifications or other conditions of the install
  - create_default_layout: Replaces slices with a set of SliceInfo objects representing a default, full partition layout
  - to_tgt: Returns a tgt.Partition representation of this partition.

2.2.2.1.2: SliceInfo

- Class Variables
  - DEFAULT_POOL: Dynamic string representing the name of the ZFS pool that will be created during installation.
- Instance Attributes
  - size: In bytes
  - offset: “Distance” in bytes from the beginning of the disk (or partition, if this slice is within a partition)
  - number
  - type: A tuple where the first value indicates the general use for this slice (e.g., UFS, ZFS) and the second value provides optional additional detail (such as ZFS pool name).
- Functions
  - is_solaris_data: Return True if this slice represents the installation target

- editable: Returns True if the user is allowed to modify the size of this slice during installation.
- adjust_offset: Adjusts the offset of this slice slightly, ensuring that it doesn't overlap with adjacent slices due to rounding errors.
- modified: Returns True if the user has modified the layout of this slice in some way
- destroyed: Returns True if the data on this slice will be destroyed as a result of modifications or other conditions of the install
- to_tgt: Returns a tgt.Slice representation of this SliceInfo

2.2.2.2: SystemInfo

- Instance Attributes
  - hostname
  - tz_region, tz_country, tz_timezone: Variables representing the desired time zone
  - time_offset: The difference between current BIOS time and the time that the clock should be set to during installation.
  - keyboard: The keyboard layout
  - locale: The default system locale
  - actual_lang: Human readable string representing the locale
- Functions
  - determine_locale: Sets “locale” and “actual_lang” based on the current system environment.

2.2.2.3: NetworkInfo

- Class Variables
  - AUTOMATIC, ~~MANUAL,~~ NONE: Variables representing possible values for “type”
  - ~~DEFAULT_NETMASK: “255.255.255.0”, the default value to use for netmask~~
  - ~~ETHER_NICS: A list of wired NICs found on the system.~~
- ~~Instance Attributes~~
  - ~~nic_name: The system's name for this NIC (e.g., “e1000g0”)~~
  - type: One of AUTOMATIC, ~~MANUAL,~~ or NONE
  - ~~ip_address: For MANUAL types, the desired IP address~~
  - ~~netmask: For MANUAL types, the netmask~~
  - ~~gateway: For MANUAL types, the default gateway, if specified~~
  - ~~dns_address: For MANUAL types, the primary DNS server address~~

- ~~domain: For MANUAL types, the DNS domain~~

- Functions
  - ~~find_dns, find_gateway, find_domain, find_netmask: Attempts to use the "dhcpinfo" command to guess what the values for those instance attributes might be~~
  - ~~get_ifconfig_data:~~
  - find_links: A static function that returns ETHER_NICS. If ETHER_NICS has not yet been set, it queries the system using dladm to determine the wired NICS on the system.

2.2.2.4: Zpool

- Instance Attributes
  - name: The zpool name
  - device: The device to create the zpool on.
  - datasets: List of ZFS Dataset Objects

2.2.2.5: ZFSDataset

- Instance Attributes
  - name: The zfs dataset name.
  - Mountpoint: The zfs dataset mountpoint.
  - be_name: Name of the boot environment.
  - zfs_swap: Boolean to indicate if a swap device is created.
  - swap_size: Size of swap.
  - zfs_dump: Boolean to indicate if a dump device is created.
  - dump_size: Size of dump.

2.2.2.6: UserInfo

- Instance Attributes
  - real_name: The real name of this user
  - login_name: The login name for this user
  - password: The hashed password for this user (passwords are hashed when this attribute is assigned)
  - passlen: An integer indicating the length of this password. Generally, 0 (not set) or 16 (password set). Does not necessarily correlate with the actual length of the password.
- Functions
  - set_password: Called when the attribute password is assigned, this encrypts the password

using install_utils.encrypt_password

2.3: Required Library Functionality

2.3.1: Overview

This section describes required library functionality that is not presently delivered in a Python API. In general, most of the disk related functionality is already available through libtd (a function will be added to libtd to gather slice inuse data from libdiskmgt). A Python bridge, libtarget_pymod, will be created, and necessary functionality exposed via that interface. In addition, a Python bridge to libzoneinfo will be created to access timezone information.

2.3.2: General Errors

The following Python errors may be thrown by any of the following functions in the described situations:

MemoryError – An underlying C function call returned NULL or otherwise was unable to allocate memory

TypeError – An argument passed in was of the wrong Python class

ValueError – An argument passed in was of the correct Python class, but the internal data could not be used

2.3.3: libtarget_pymod

2.3.3.1: discover_target_data

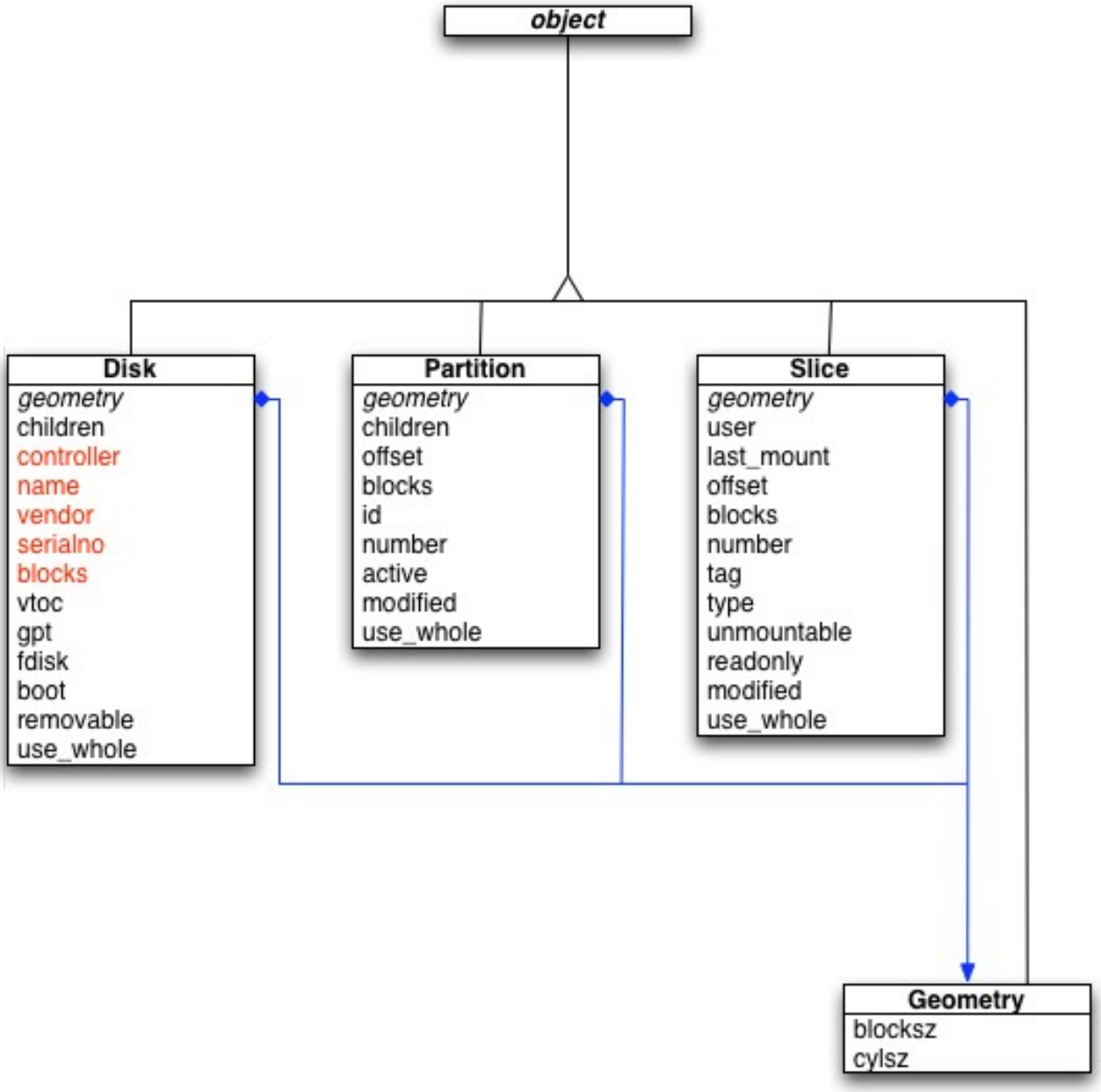
This is the factory function that discovers all the physical targets on a system. It makes all the necessary calls to libtd.so.

2.3.3.1.1: Input

None

2.3.3.1.2: Output

List of tgt.Disks, each of which may have children that are either tgt.Partition or tgt.Slice. If a tgt.Disk's children are a tgt.Partition it may have tgt.Slice children.



All Python type extensions must inherit from *object*, the builtin base class. tgt.Disk, tgt.Partiton, tgt.Slice, and tgt.Geometry are all Python type extensions. They look like very simple Python objects.

The properties in red on tgt.Disk are read-only. They are set by __init__() and can't be changed.

The *children* of tgt.Disk is a tuple of either tgt.Partition or tgt.Slice. The *children* of tgt.Partition is a tuple of tgt.Slice.

Everything returned, the list of tgt.Disk, its children, and its grandchildren (if it has any) all point to the same tgt.Geometry *object*. They share it so that calculations can be made in human readable units.

2.3.3.1.3: Errors

The following exce may be raised:

MemoryError – the C bridge ran out of memory.

tgt.TgtError – raised for any other error in libtd.so. A string is set with the description of the error.

2.3.3.1.4: Details

This function will initiate target discovery. It will discover disks, partitions and slices on the system.

Currently, liborchestrator contains the logic that manages both callbacks, and explicitly iterating through discovery of each item type. This will shift into libtd.

2.3.3.2: create_disk_target

2.3.3.2.1: Input

A python object containing the arguments. Only 1 argument is passed and that is the disk object.

2.3.3.2.2: Errors

The following may be raised:

tgt.TgtError – raised for any other error in libti.so. A string is set with the descripton of the error.

2.3.3.2.3: Details

This function will detect if the system is a sparc or x86 system and create a fdisk or vtoc target accordingly using libti. If it is a GPT labelel disk on sparc, it will relabel it as SMI.

2.3.3.3: create_zfs_root_pool

2.3.3.3.1: Input

A python object containing the arguments. Only 1 argument is passed and that is the TgtZpool object for the zpool to be created.

2.3.3.3.2: Errors

The following may be raised:

tgt.TgtError – raised for any other error in libti.so. A string is set with the descripton of the error.

2.3.3.3.3: Details

This function, given the information needed for a zfs root pool, will create the appropriate pool via the libti library.

2.3.3.4: create_zfs_volume

2.3.3.4.1: Input

A python object containing the arguments. Five arguments are passed, the root pool name for the volume, a boolean indicating if zfs swap should be created, the size of the swap to create, a boolean indicating if zfs dump should be created, and the size of the dump to create.

2.3.3.4.2: Errors

The following may be raised:

tgt.TgtError – raised for any other error in libti.so. A string is set with the descripton of the error.

2.3.3.4.3: Details

This function will create a zfs volume on the zfs pool specified. If indicated, the volume will be set as swap and/or dump. These operations are performed via a call to the libti library.

2.3.3.5: create_be_target

2.3.3.5.1: Input

A python object containing the arguments. Four arguments are passed, the root pool name for the be, the name for the be, the installed root directory, and a tuple with the names of the shared file systems for the be.

2.3.3.5.2: Errors

The following may be raised:

tgt.TgtError – raised for any other error in libti.so. A string is set with the descripton of the error.

2.3.3.5.3: Details

This function will attempt to create a BE via the libti library.

2.3.3.6: release_zfs_root_pool

2.3.3.6.1: Input

A python object containing the arguments. One arguments is passed in, the name of the zfs root pool to be released.

2.3.3.6.2: Errors

The following may be raised:

tgt.TgtError – raised for any other error in libti.so. A string is set with the descripton of the error.

2.3.3.6.3: Details

This function will call the libti library to release the indicated zfs root pool.

2.3.3.7: retrieve_tgt_utils_module

The purpose of this function is to provide the C extension module, *tgt*, with a Python function.

This is used by the standard method *__str__()* for tgt.Disk, tgt.Partition, and tgt.Slice.

Because string processing is so much easier in Python than in C, the *__str__()* method simply calls whatever *callable* was passed in giving it the *self* argument and a string identifying its type.

This function should be called after importing the *tgt* module and before calling *discover_target_data* (see section 2.3.3.1) or attempting to print a *tgt.Disk*, *tgt.Partition*, or *tgt.Sice*.

2.3.3.7.1: Input

A Python *callable* object, probably a function.

The signature of this *callable* should receive two arguments: a Python *object* and a Python string.

It must also return a string.

While that is sufficient to prevent core dumps to simply return the empty string, for useful debugging the callable should switch on the python string passed in and provide debugging information about the *tgt* object.

2.3.3.7.2: Errors

If the callable does not accept the correct arguments and return a string, any code attempting to print a *tgt.Disk*, *tgt.Partition*, or *tgt.Slice* will fail and potentially core dump or cause Python to exit with a stack trace.

2.3.3.7.3: Details

2.3.4: libtd

2.3.4.1: ddm_get_slice_inuse_stats

2.3.4.1.1: Input

Name of slice

nv attribute list to which inuse data is added

2.3.4.1.2: Output

0 – finished successfully

error code - failed

2.3.4.1.3: Details

Calls libdiskmgt to get the inuse data for a particular slice and add data to passed in nv attribute list

2.3.5: libzoneinfo_pymod

2.3.5.1: get_tz_info (libzoneinfo_pymod)

2.3.5.1.1: Input

Python object containing 0,1, or 2 args

Number of args determines type of information returned

0 args - returns continent info

1 arg (ctnt_name) - returns country info

2 args (ctnt_name, ctry_code) - returns timezone info

2.3.5.1.2: Output

On success: pylist of tuples, each tuple has three elements:

tz_name: names of continent, country, or timezone

tz_descr: descriptive name of continent, country, or timezone

tz_loc: localized name of continent, country, or timezone

On failure: empty pylist (or memory error if unable to create pylist)

2.3.5.1.3: Details

Calls:

libzoneinfo:get_tz_continents,

libzoneinfo:get_tz_countries, and

libzoneinfo:get_timezones_by_country

to obtain timezone information.

2.3.5.2: tz_isvalid (libzoneinfo_pymod)

2.3.5.2.1: Input

Name of timezone to be checked for validity.

2.3.5.2.2: Output

Returns results of call to isvalid_tz (1 if timezone is valid).

2.3.5.2.3: Details

Makes call into libzoneinfo:isvalid_tz to determine validity of timezone.

2.4: Non-Library Functionality

The following major functions will be written for and used by the Text Installer. In general, they are considered to be private functions.

2.4.1: General Errors

The following Python errors may be thrown by any of the following functions in the described situations:

TypeError – An argument passed in was of the wrong Python class

ValueError – An argument passed in was of the correct Python class, but the internal data could not be used

InstallationError - Some sort of error occurred during installation. This error just indicates that something is wrong. The exact cause of the error will be recorded in the log file.

NotEnoughSpaceError - There is not enough space in the target disk for successful installation.

2.4.2: Class: SwapDump

2.4.2.1: Summary

A class representing all information associated with determining the amount of swap and dump space for the installation.

2.4.2.2: Functions:

2.4.2.2.1: __init__

Initialize swap/dump calculation with the size of the running system's memory.

2.4.2.2.2: get_required_swap_size

Determines whether swap is required. If so, the amount of space used for swap is returned. If swap is not required, 0 will be returned. Value returned is in MB. If system memory is less than 900mb, swap is required. Minimum required space for swap is 0.5G.

2.4.2.2.3: calc_swap_dump_size

2.4.2.2.3.1 Input

- installation_size: size required for installation (MB)
- available_size: available size in the target disk. (MB)

- swap_included: Indicate whether required swap space is already included and validated in the installation size. Default to false.

2.4.2.2.3.2 Output

A tuple: (swap_type, swap_size, dump_type, dump_size)

2.4.2.2.3.3 Details

Calculate swap/dump, based on the amount of system memory, installation size and available size.

The following rules are used for determining the type of swap to be created, whether swap zvol is required and the size of swap to be created.

Memory	Type	Required to create	size
<900mb	Zvol	Yes	0.5G (MIN_SWAP_SIZE)
900mb-1G	Zvol	No	0.5G (MIN_SWAP_SIZE)
1G-64G	Zvol	No	0.5G-32G (half of memory)
>64G	Zvol	No	32G (MAX_SWAP_SIZE)

The following rules are used for calculating the amount of required space for Dump.

Memory	Type	size
<0.5G	Zvol	256MB (MIN_DUMP_SIZE)
0.5-32G	Zvol	256MB-16G (half of memory)
>32G	Zvol	16G (MAX_DUMP_SIZE)

If slice/zvol is required, and there's not enough space in the target, an error will be raised. If swap zvol is not required, and there's not enough space in the target, as much space as available will be utilized for swap/dump . All calculations for size are done in MB.

2.4.2.2.4: __calc_size

2.4.2.2.4.1 Input

- available_swap_space: space that can be dedicated to swap in MB
- min_size: minimum size to use
- max_size: maximum size to use

2.4.2.2.4.2 Output:

size of swap in MB

2.4.2.2.4.3 Details

Calculates size of swap or dump in MB based on amount of physical memory available.

If less than calculated space is available, swap/dump size will be trimmed down to the avaiable space. If calculated space is more than the max size to be used, the swap/dump size will be trimmed down to the maximum size to be used for swap/dump

2.4.2.2.5: get_swap_device

Return the string representing the device used for swap.

2.4.3: Function: get_image_size

2.4.3.1: Input

None

2.4.3.2: Output

Size retrieved from the .image_info file in MB.

2.4.3.3: Details

Total size of the software in the image is stored in the /.cdrom/.image_info indicated by the keywoard IMAGE_SIZE. This function retrieves that value from the .image_file . The size recorded in the .image_file is in KB, other functions in this file uses the value in MB, so, this function will return the size in MB

2.4.4: Function: get_system_memory

2.4.4.1: Input

None

2.4.4.2: Output

Returns the amount of memory available in the system. The value returned is in MB.

2.4.5: Function: save_timezone_in_init

2.4.5.1: Input:

- basedir: Alternate root used for the installation.
- timezone: Timezone string to be stored.

2.4.5.2: Output:

None

2.4.5.3: Details:

Save the timezone in <basedir>/etc/default/init

2.4.6: Function: get_minimum_size

2.4.6.1: Input

A SwapDump object containing size of the swap and dump zvol to be created for the installation.

2.4.6.2: Output

Returns the minimum amount of space required to perform an installation.

2.4.6.3: Details

The amount of space returned by this function includes the value that's retrieved from the .image_info file in the booted media, a predefined amount of overhead space, and the minimum amount of space required for creating the swap zvol for low-memory systems.

2.4.7: Function: get_recommended_size

2.4.7.1: Input

A SwapDump object containing size of the swap and dump zvol to be created for the installation.

2.4.7.2: Output

Returns the recommended installation size in MB.

2.4.7.3: Details

The amount of space returned by this function includes the value that's retrieved from the .image_info file in the booted media, a predefined amount of overhead space, and the minimum amount of space for creating the swap zvol, plus the estimated space to perform an upgrade.

2.4.8: Function: setup_etc_vfstab_for_swap

2.4.8.1: Input

- swap_device: string representing the swap device
- basedir: Alternate root used for the installation.

2.4.8.2: Output

None

2.4.8.3: Details

Add the swap device to /etc/vfstab.

2.4.9: Function: perform_ti_install

2.4.9.1: Input

All input values will be passed directly into do_ti_install().

- An InstallationProfile object
- screen object to be used by the update_status_function
- Update status function
- quit_event
- time_change_event

2.4.9.2: Ouput

None

2.4.9.3: Detail

Wrapper to call the do_ti_install() function. Sets the install_succeeded in the InstallationProfile variable indicating whether the installation is successful or not.

2.4.10: Function: do_ti_install

2.4.10.1: Input

- An InstallationProfile object
- screen object to be used by the update_status_function
- Update status function
- quit_event
- time_change_event

2.4.10.2: Ouput

None

2.4.10.3: Errors

InstallationError – Thrown for any error occurred during install. Cause of error is recorded in the log file.

2.4.10.4: Detail

Installation engine for the text installer.

This function performs the following operations:

1. Validates all values required for installation specified the Installation Profile.
2. Calculates the amount of swap and dump based on the amount of memory in the system.
3. Sets the system's time based on timezone information provided by the user.
4. Instantiates installation targets using the do_ti() function.
5. Copies the bits from the installation media to the target system by calling the do_transfer() function.
6. Calls run_ICTs() function to customize the installed system.
7. Run post_install_cleanup() to umount all targets mounted for installation and transfer the log file to the installed system.

2.4.11: Class: InstallStatus

2.4.11.1: Summary

Stores information on installation progress and provide a hook for updating the screen. The actual install is viewed as 3 components from the progress reporting point of view. Target Instantiation, Transfer, and ICT. The InstallStatus class assigns a weight to each of the components. Each components reports their progress as percentage completed to the InstallStatus class. The percentage completed is re-calculated based on the weight assigned to the different components. The weighted progress is displayed in the UI.

2.4.11.2: Functions

2.4.11.2.1: __init__

2.4.11.2.1.1 Input

screen: value passed in from the UI

update_status_function: value passed in from UI.

quit_event: value passed in from UI.

2.4.11.2.1.2 Output

None

2.4.11.2.1.3 Details:

Initialize relative ratio used for computing the overall progress of the installation. The following assumption is made:

Target Instantiation: 5% of the overall progress

Transfer of bits: 93% of the overall progress

ICT: 2% of the overall progress

2.4.11.2.2: update

Calculates the weighted progress based on progress information provided by different components, and updates the install status by calling the update_status_function provided by the UI. Also checks the quit_event to see if the installation should be aborted.

2.4.12: Function: do_ti

2.4.12.1: Input

- Installation Profile object
- SwapDump object

2.4.12.2: Output

None

2.4.12.3: Errors

InstallationError

2.4.12.4: Details

This function calls the Python bridge to libti.so to create the disk layout, create a zfs root pool, create zfs volumnes for swap and dump, and create a BE.

2.4.13: Function: do_transfer

2.4.13.1: Input

None

2.4.13.2: Output

None

2.4.13.3: Errors

InstallationError

2.4.13.4: Details

This function calls the tm_perform_transfer() function in libtransfer to perform a CPIO transfer of bits from the media to the target.

2.4.14: Function: run_ICTs

2.4.14.1: Input

An InstallationProfile object

2.4.14.2: Output

None. Throws exceptions if any stage fails fatally.

2.4.14.3: Errors

ICTError (extends InstallationError: see perform_install) – Used internally if an ICT returns an error code that indicates a failure; cause of the failure is then logged if needed. Additionally, if the error is non-recoverable, then the error is thrown higher.

2.4.14.4: Details

This is a wrapper for running all needed ICTs. For each ICT, the necessary data will be pulled out of the InstallationProfile and passed to the ICT in the required format. After each ICT, a call will be made to update_install_status.

The following Python ICTs will be called via the /sbin/install-finish script.

- create_smf_repository
- configure_network
- create_mnttab (x86)
- cleanup_unneeded_files_and_directories
- keyboard_layout
- delete_misc_trees
- set_prop_from_eeprom (x86)
- set_console_boot_device_property (x86)
- set_Solaris_partition_active
- remove_bootpath (x86)
- fix_grub_entry (x86)
- add_other_OS_to_grub_menu (x86)
- update_dumpadm_nodename

- explicit_bootfs (x86)
- update_boot_archive
- create_sparc_boot_menu (SPARC)
- copy_sparc_bootlst (SPARC)
- remove_files
- copy_splash_xpm (x86)
- smf_correct_sys_profile
- add_sysidtool_sys_unconfig_entries
- remove_liveCD_environment
- remove_install_specific_packages
- set_flush_content_cache_on_success_false
- set_root_password
- create_new_user
- reset_image_UUID

The following are C based ICTs. They are called via the /opt/install-test/bin/ict_test program.

- ict_set_lang_locale
- ict_configure_user_directory
- ict_set_host_node_name
- ict_set_user_profile
- ict_installboot
- ict_set_user_role
- ict_snapshot
- ict_mark_root_pool_ready

2.5: Extended/Logical Partition Manipulation

2.5.1: Overview

Extended partitions will be managed very similarly to standard partitions. Only a single extended partition can be created, and up to 32 logical partitions can be allocated within that extended partition. On the partition selection screen, logical partitions will be presented in a separate list, next to standard partitions. However, manipulation of both standard and logical partitions is much the same. See the UI spec for specific details on how standard, extended and logical partitions work.

http://www.opensolaris.org/os/project/caiman/TextInstallerProject/UI_Specification/

2.6: Slice Manipulation

2.6.1: Overview

Slice management will be very similar to partition management. A user will be able to create, delete and edit the size of slices, up to the maximum number of slices available. Again, see the UI spec for specific details on how slice manipulation will function.

http://www.opensolaris.org/os/project/caiman/TextInstallerProject/UI_Specification/

2.6.2: Root Pool Name

Slices will be preserved unless explicitly removed or overwritten. The root pool name will not be hard-coded to “rpool”, but may also be called “rpool1”, “rpool2”, etc., to allow for co-existence with existing zpools.

2.7: New Text Installer Media

2.7.1: Overview

The Text Installer will be delivered as a new media image. This image will be built using the Distribution Constructor, and will contain a standard set of packages targeted at basic server installations (see “Server Software Set” below). In order to provide a consistent experience across platforms (x86 and SPARC), on x86 machines there will only be one GRUB entry, and that entry will be booted quickly. A timeout of 5 seconds will be set, and the menu.lst file will enable the “hiddenmenu” option, which hides the GRUB menu from users unless they hit Esc. During the boot sequence, the user will be asked for language and keyboard information, similar to the liveCD boot sequence.

At this point, the user is then asked to choose from a small set of choices. The list items delivered as part of the Text Installer project will be “Install OpenSolaris”, “Shell”, “Terminal type”, and “Reboot”. Choosing “Install OpenSolaris” will begin the Text Installer executable; choosing “Shell” will drop the user to a prompt. In both cases, upon exiting, the user is again presented with the menu of choices. This allows the user to, for example, begin the Text Installer, notice an error, quit and choose to start the shell prompt, diagnose and potentially fix the problem, and restart the Text Installer, all without rebooting. Similarly, the user could exit the Text Installer to change the terminal type if display issues were encountered and then restart the Text Installer. Note that the user could also run /usr/bin/text-install directly, if so desired.

While the Text Installer media will be a separate image from the liveCD, the layout of both media will be similar. This will allow use of the transfer module “as is,” since that library makes assumptions about the layout of the media from which the installation is performed. An iso sort file will need to be generated, similar to the liveCD, to optimize boot time.

2.7.2: SMF Changes

A new service will be created for the text installer, system/install-setup:default, which will display the small menu of choices. This service will have a dependency on milestone/single-user. That is, once milestone/single-user is online, system/install-setup will be brought online and the small menu of choices displayed.

The filesystem/root:media service will be changed to detect the text install environment and subsequently invoke the keyboard and language selection prompts.

2.7.3: Distribution Constructor Changes

The new image will be built using the Distribution Constructor. As part of the delivery for the Text Installer, the following new files will be added to the install/distribution-constructor package:

text_mode_x86.xml

text_mode_sparc.xml

text_live.xml

text_install_x86_iso.sort

tm_pre_boot_archive_pkg_image_mod

The first two files, text_mode_x86.xml and text_mode_sparc.xml, will be manifests that build the default Text Installer image. These manifests will differ from the liveCD manifest in four primary areas.

a) The package list will include the server software set, as described below. Gnome will be absent (as will the majority of other desktop utilities), and the Text Install package will be added.

b) It will point the br-config DC step to text_live.xml. This file describes the services needed to boot the Text Installer media to an appropriate environment. In particular, nwam/hal/rmvolmgr will be enabled, console-login will be disabled, gdm services will be absent, and the text-installer service will be enabled.

c) For x86, the GRUB menu will be hidden (via the 'hiddenmenu' GRUB command). This means that the Text Installer user experience between x86 and SPARC will be largely the same, even during booting.

d) For x86, the text_install_x86_iso.sort file will be specified as the iso_sort file.

Additionally, the SMF services that are common across the AI, LiveCD and Text Install media will be extracted out into a common file, generic_live.xml that will then be referenced in each of the specific media profile files, for example text_live.xml. This is expected to alleviate some of the maintenance burden that comes with maintaining a number of files that reference the same/similar services.

2.7.4: Additional Packages

In addition to the basic packages required to run OpenSolaris and the extra software delivered as part of the “Server Software Set,” the following packages will be on the media:

- system/install/text-install
  - Contains the executable, /usr/bin/text-install, and all needed files
  - Dependencies on runtime/python-26, system/install, system/library/install
- library/ncurses
  - The Python curses module is linked against this package as of build 124 – meaning system/install/text-install depends on runtime/python-26, which depends on library/ncurses
- shell/expect
  - The “expect” program is provided to simplify automated testing.

3: External Components

3.1: Python Curses Module

Python will be used as the Text Installer's programming language. The curses module of Python will be used to create the screens. It is important to note that due to some issues with the standard curses library, the Python curses module is linked against the nCurses library, libncurses, in order to deliver the needed functionality. See CR 6872653.

Note that “Python curses module”, “curses library” and “nCurses library” are three different things. The term “curses library” refers to the initial, older implementation of a C library of common terminal functions. The term “nCurses library” refers to a newer implementation of the same C library, with a few additional features and better stability on recent hardware and software. The term “Python curses module” refers to the Python module which delivers the functionality provided by either the curses library or the nCurses library.

3.2: Extended Partition Library

Extended partition manipulation depends on the delivery of updates to libtd (see bug 1777, http://defect.opensolaris.org/bz/show_bug.cgi?id=1777) and the new library allowing the discovery, manipulation, and validation of extended partitions and logical partitions within the extended partition.

~~3.3: Static IP Configuration Interface~~

~~The networking team will, in the future, deliver a standard interface for configuring an OpenSolaris machine with a static IP. This interface is expected to work for both installed systems, as well as installer environments. See configure_network section above on Static IP Configuration. Also see bug 10307, http://defect.opensolaris.org/bz/show_bug.cgi?id=10307. Once the interface is delivered, Static IP configuration will be added to the Text Installer.~~

3.4: Server Software Set

As the Text Installer will be targeted at installations of server machines, it will deliver a set of software

that comprises a basic headless system (implicitly no Gnome desktop). Examples of software included are: ZFS, IPS, Zones, Dtrace, Dtrace toolkit and Automated Installer.

4: Phased Work

Generally, the Text Installer work can be divided up into three categories:

1. Creation of new media
    - Distribution Constructor updates to output the new media
    - Verification of the package list as stable
    - Development of a booted environment suitable for running the Text Installer executable
2. Bare Minimum to perform installation
    - The minimum amount of screens and interactive questions needed to perform an installation on par with the installation experience provided by the liveCD GUI Installer today
3. New Feature Implementation
    - Extended Partition manipulation
    - Slice manipulation and preservation
    - Static IP Configuration

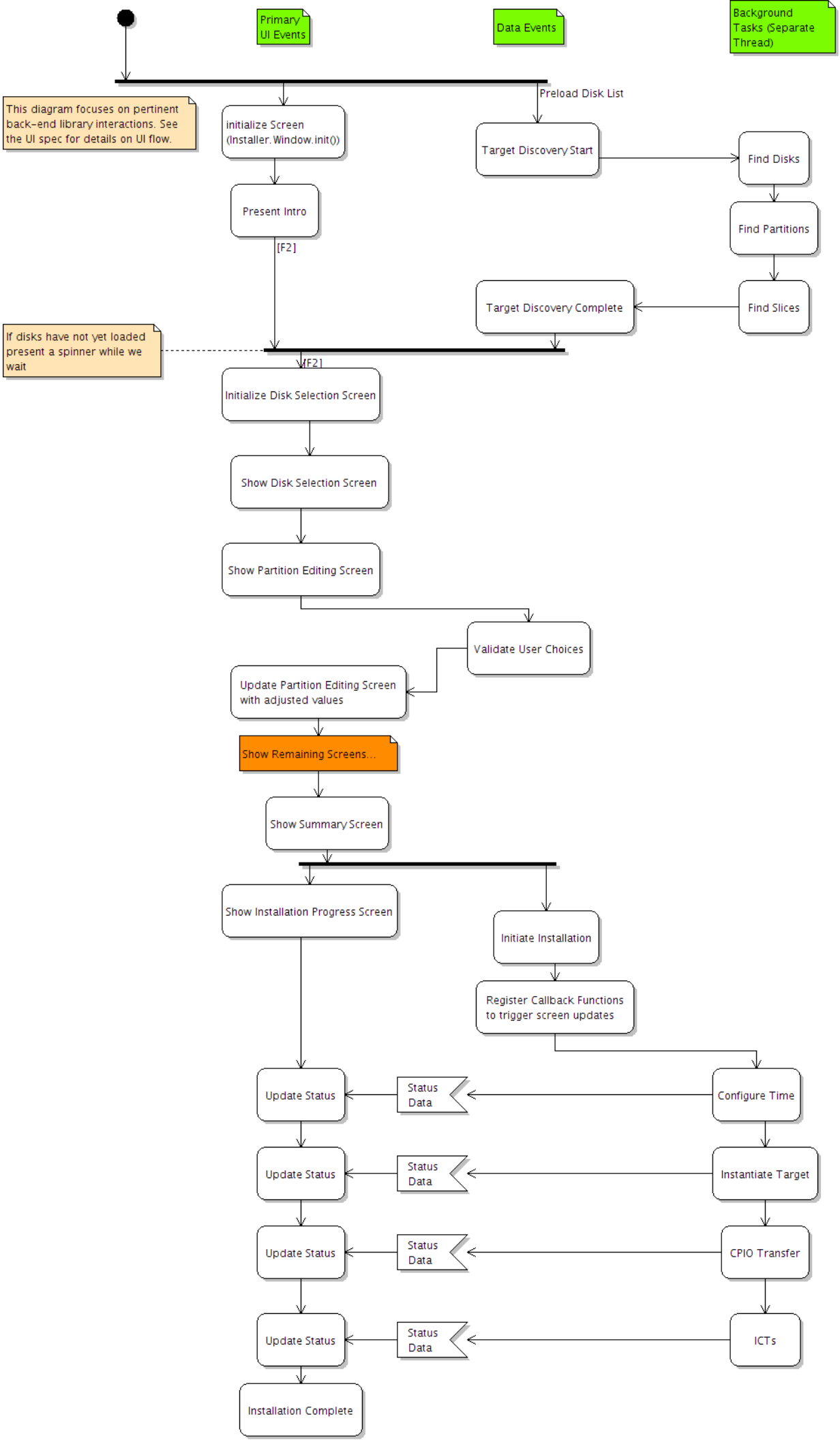
Items 1 and 2 are considered absolutely required for delivery of a Text Installer to be meaningful – that is, perform a stable installation. Item number 3 is needed before the Text Installer will be considered "feature complete" and satisfy the needs of its target audience.

5: CUD Considerations

In order to fit in with the intent of the Caiman Unified Design, the Text Installer intends to manage its own library calls and user selections. As such, liborchestrator will not be used; instead, the Text Installer will directly call into libtd, libti, libtransfer and libict. As libtd does not currently have a Python bridge, one will be created as part of this work. Finally, any functionality that liborchestrator delivers that is critical for the Text Installer will be implemented in Python using the InstallProfile set of classes (in particular, the liborchestrator function om_validate_and_resize_partition_info). This approach is taken, instead of attempting to use calls into liborchestrator, as liborchestrator's tight coupling with the GUI Installer makes use of those functions non-ideal.

6: Appendix Diagram: Data Flow

The diagram on the following pages illustrates the main library interactions: target discovery, disk partition validation, and installation flow. Note that slice validation is left out to save space, but would work similarly to partition validation.


Primary UI Events
Data Events
Background Tasks (Separate Thread)
This diagram focuses on pertinent back-end library interactions. See the UI spec for details on UI flow.
initialize Screen (Installer.Window.init())
Preload Disk List
Target Discovery Start
Find Disks
Find Partitions
Find Slices
Target Discovery Complete
Present Intro
[F2]
If disks have not yet loaded present a spinner while we wait
[F2]
Initialize Disk Selection Screen
Show Disk Selection Screen
Show Partition Editing Screen
Validate User Choices
Update Partition Editing Screen with adjusted values
Show Remaining Screens...
Show Summary Screen
Show Installation Progress Screen
Initiate Installation
Register Callback Functions to trigger screen updates
Update Status
Status Data
Configure Time
Update Status
Status Data
Instantiate Target
Update Status
Status Data
CPIO Transfer
Update Status
Status Data
ICTs
Installation Complete