sdi::filesystem Namespace Reference

Data Structures	
struct	UpdateFiles

Enumerations	

enum	Location {
	
	LOC_None,
	LOC_SdiFlashDir,
	LOC_SdiConfigDir,
	LOC_SdiExtConfigDir,
	
	LOC_EmvFlashDir,
	LOC_EmvConfigDir,
	LOC_LogConfigDir,
	LOC_NfcFlashDir,
	
	LOC_SdiExtFontsDir
	
	}

enum	Action {
	
	ACT_None = 0,
	ACT_SecInit = (1 << 0),
	ACT_DisableEpp = (1 << 1),
	ACT_EmvExit = (1 << 2),
	
	ACT_EmvExitComplete = (1 << 3),
	ACT_EmvFlashPerm = (1 << 4),
	ACT_EmvInitInfo = (1 << 5),
	ACT_LogInit = (1 << 6),
	
	ACT_CardRanges = (1 << 7),
	ACT_WhitelistFile = (1 << 8),
	ACT_AclInit = (1 << 9)
	
	}

enum	ActionSource { ASRC_Default = 0,
	ASRC_Removal = (1 << 0),
	ASRC_PostAction = (1 << 1)
	}

enum	FilesModes { FM_Default = 0,
	FM_NoAbort = 1,
	FM_Quiet = 2,
	FM_KeepTopDir = 4
	}

Functions

void	factory_reset ()
void	init (enum config::SdiSysConfig::SDIMode sdi_mode)
bool	read_file (const char *file, string &data)
bool	write_file (const char *file, const string &data)
bool	copy_file (const string &src, const string &dest)
bool	move_file (const string &src, const string &dst)
int	get_dir_files (const char *dir, vector< string > *files, const char *regex)
int	copy_files (const vector< string > &files, const char *dst_dir, unsigned modes)
int	remove_dir (const string &path, unsigned modes)
int	remove_files (const vector< string > &files, unsigned modes)
const char *	binary_dir ()
string	home_flash_file (const string &file)
string	home_config_file (const string &file)
string	lookup_config_file (const string &file)
const char *	plugin_config_dir ()
int	extract_tar (const string &tarfile, const string &destdir, const char *regex)

unsigned short	install_emv_config_package (const string &emv_config_pkg)
const char *	home_flash_dir ()
const char *	home_config_dir ()
const char *	ext_config_dir ()
const char *	emv_flash_dir ()
const char *	emv_config_dir ()
const char *	log_config_dir ()
const char *	nfc_flash_dir ()
const char *	home_lib_dir ()
const char *	ext_plugin_dir ()
const char *	ext_font_dir ()
const char *	tmp_dir ()
const char *	upload_install_dir (bool flash)
const char *	sys_remove_sponsor_dir ()
const char *	ccp_resource_dir ()
const char *	ccp_database_dir ()
const char *	sdi_update_dir ()
const char *	sdi_persist_dir (bool system)
const char *	sdi_certstore_dir ()
unsigned short	install_user_configuration (bool recover)
unsigned short	remove_user_configuration ()
unsigned short	install_sdi_plugins (bool recover)
unsigned short	remove_sdi_plugins ()
bool	read_file (const char *file, std::string &data)
bool	write_file (const char *file, const std::string &data)
bool	copy_file (const std::string &src, const std::string &dest)

bool	move_file (const std::string &src, const std::string &dest)
int	get_dir_files (const char *dir, std::vector< std::string > *files=0, const char *regex=0)
int	copy_files (const std::vector< std::string > &files, const char *dst_dir, unsigned modes=FM_Default)
int	remove_files (const std::vector< std::string > &files, unsigned modes=FM_Default)
int	remove_dir (const std::string &path, unsigned modes=FM_Default)
int	extract_tar (const std::string &tarfile, const std::string &destdir, const char *regex=0)
std::string	home_flash_file (const std::string &file)
std::string	home_config_file (const std::string &file)
std::string	lookup_config_file (const std::string &file)
unsigned short	install_emv_config_package (const std::string &emv_config_pkg)

Variables

const struct UpdateFiles	allowed_usr_files []
const unsigned	allowed_usr_files_size = (sizeof(allowed_usr_files) / sizeof(allowed_usr_files[0]))

Data Structure Documentation

◆ sdi::filesystem::UpdateFiles

struct sdi::filesystem::UpdateFiles

update file definition to check for allowed files to update and remove

Data Fields		
enum Location	dest	destination on terminal, see enum Dest

const char *	dest_prefix	file prefix (path) used in application folder (if any)
const char *	file	name of the update file (without path)
unsigned long	post_actions	post-actions (bitmask) to execute for this file after update (see enum Action)
unsigned long	pre_actions	pre-actions (bitmask) to execute for this file before update (see enum Action)
const char *	prefix	file prefix (path) in the user configuration package
bool	regex	file is already a regular expression

Enumeration Type Documentation

◆ Action

enum Action

actions supported by run_actions() below (bitmask used to combine actions). Order of execution is sprecified by run_actions() and recently corresponds to enum order.

Enumerator	
ACT_None	no action
ACT_SecInit	call secInit() for this file (Android only)
ACT_DisableEpp	disable attached EPP (if enabled as Countertop)
ACT_EmvExit	call EMV_CT_Exit_Framework() and EMV_CTLS_Exit_Framework() for this file
ACT_EmvExitComplete	call EMV_CT_Exit_Framework_extended(EXIT_CT_COMPLETE) and EMV_CTLS_Exit_Framework_extended(EXIT_CTLS_COMPLETE) for this file

ACT_EmvFlashPerm	call set_emv_flash_permissions() on EMV flash config folder (VOS/VOS2/VOS3 only) Note: ACT_EmvFlashPerm must be below ACT_EmvInitInfo so that it is executed before ACT_EmvInitInfo as both are used together in install_emv_config_package()
ACT_EmvInitInfo	call initEmvInfo() to rebuild EMV Info cache
ACT_LogInit	call LogAPI_ReconfigNotification() for this file
ACT_CardRanges	reload cache for card ranges configuration
ACT_WhitelistFile	reload cache for whitelist configuration
ACT_AclInit	reload ACL (access control list) file

◆ ActionSource

enum ActionSource

action source specifying for what run_actions() was invoked

Enumerator	
ASRC_Default	default: installation (see ASRC_Removal) and pre-action (see ASRC_PostAction)
ASRC_Removal	removal (not installation)
ASRC_PostAction	post-action (not pre-action)

◆ FilesModes

enum FilesModes

modes for functions copy_files(), remove_files(), remove_dir()

Enumerator	

FM_Default	let first file error abort the function with -1.
FM_NoAbort	function proceeds, in case of single files cannot be copied or removed. Function returns with number of successfully pocessed files.
FM_Quiet	don't issue information to log (errors only), e.g. for non-existing files
FM_KeepTopDir	only supported for remove_dir(): keep top level directory, after the content was removed

◆ Location

enum Location

file location directory for SDI configuration files

Enumerator	
LOC_None	No directory
LOC_SdiFlashDir	SDI flash configuration directory (read/write)
LOC_SdiConfigDir	SDI internal configuration directory (read-only)
LOC_SdiExtConfigDir	SDI directory for external user configuration (read-only)
LOC_EmvFlashDir	EMV flash configuration directory (read/write)
LOC_EmvConfigDir	EMV configuration directory (read-only)
LOC_LogConfigDir	ADK Logging configuration directory (read-only)
LOC_NfcFlashDir	ADK NFC configuration directory (read/write)
LOC_SdiExtFontsDir	SDI directory for external fonts (installed with user config package (read-only)

Function Documentation

◆ binary_dir()

const char * binary_dir	(		)	

return the absolute path to the SDI server binary (without filename)

Returns
absolute path to SDI server binary

On Titus, this function returns an empty string.

◆ ccp_database_dir()

const char * ccp_database_dir	(		)	

return absolute path to database folder of CCP (ADK COM CONTROL PANEL) (VOS/VOS2/VOS3: $HOME/flash/com) (Android: NULL, no CCP support)

This function returns NULL, if the platform has no CCP support.

Returns
absolute path to CCP resource directory or NULL in case of error/no CCP support

◆ ccp_resource_dir()

const char * ccp_resource_dir	(		)	

return absolute path to resource folder of CCP (ADK COM CONTROL PANEL) (VOS/VOS2/VOS3: <ext_cfg_dir>/ccp/www or $HOME/www/ccp or NULL, if no resources are found. <ext_cfg_dir> as provided by ext_config_dir()) (Android: NULL, no CCP support)

This function returns NULL, if the platform has no CCP support.

Returns

absolute path to CCP resource directory or NULL in case of error/no CCP support

◆ copy_file() [1/2]

bool sdi::filesystem::copy_file	(	const std::string &	*src*,
		const std::string &	*dest*
	)		

helper function to copy a file from source `src` to destination `dest`.

Parameters

[in]	src	source file location
[in]	dest	destination file location

Returns

true for success, else false for error

◆ copy_file() [2/2]

bool sdi::filesystem::copy_file	(	const string &	*src*,
		const string &	*dest*
	)		

◆ copy_files() [1/2]

int sdi::filesystem::copy_files	(	const std::vector< std::string > &	*files*,
		const char *	*dst_dir*,
		unsigned	*modes* = `FM_Default`

	)		

helper function to copy multiple files to destination directory `dst_dir`. Source files to be copied are specified by vector `files`.

Parameters

[in]	files	vector of files to be copied to destination directory
[in]	dst_dir	path to destination directory, where the files are copied to
[in]	modes	function operation modes, see enum FilesModes

Returns

number of files, which were copied. -1 is returned in case of error due to missing directory or file access.

◆ copy_files() [2/2]

int sdi::filesystem::copy_files	(	const vector< string > &	*files*,
		const char *	*dst_dir*,
		unsigned	*modes*
	)		

◆ emv_config_dir()

const char * emv_config_dir	(		)	

return the absolute path to folder for read-only EMV files. (VOS/VOS2: /etc/config/adkemv) (Android: $HOME) (VOS3: <ext_cfg_dir>/emv; <ext_cfg_dir> as provided by ext_config_dir()) (Titus: $HOME, with /sdi as $HOME, even emv-desired.xml is not supported on this platform)

Required for SW update on Android. On VOS and Titus, this function is unused.

Returns

absolute path to HOME for read-only EMV files.

◆ emv_flash_dir()

const char * emv_flash_dir	(		)	

return the absolute path to flash folder for writeable EMV files. If the directory does not exist, it is created and group is adjusted for EMV usage. (VOS/VOS2: /mnt/flash/etc/config/adkemv) (VOS3: /mnt/appdata/versioned/globalshare/sdi/emv) (Android: $HOME/flash/adkemv) (Titus: $HOME/flash/adkemv, with /sdi as $HOME)

Also required for SW update on Android and to access "EMV_Terminal.xml" to get terminal language.

Returns
absolute path to shared flash folder for writeable EMV files

◆ ext_config_dir()

const char * ext_config_dir	(		)	

return the abloslute path of external SDI configuration folder directory. (VOS/VOS2: /etc/config/sdi) (VOS3: /home/usr<X>/sdi) (Android: $HOME/sdi_ext) (Titus: $HOME/ext, with /sdi as $HOME) This folder holds the external configuration files, which were installed by user configuration package to overload SDI default configuration. The external SDI configuration folder contains files for read access only.

Returns
absolute path to external SDI configuration folder (with read-only files)

On VOS3 SDI checks at startup for the existence of the folder /home/usr<X>/sdi starting from usr1 up to usr16. If the folder is found under a specific user, this user directory is applied for external SDI configuration. This means, the first user providing the folder wins, other users with higher user numbers will be ignored.

◆ ext_font_dir()

const char * ext_font_dir	(		)	

return the absolute path of the folder for external fonts. (Android: $HOME/fonts/sdi_ext) (VOS/VOS2/VOS3: /usr/share/fonts) (Titus: <ext_cfg_dir>/fonts; <ext_cfg_dir> as provided by ext_config_dir(), unused so far)

> This function is un-used on VOS, since fonts are installed with SI font packages to the system folder, we ADK components have access to use them. On Android, ADKPRT runs in context of SDI, therefore, fonts are installed with an user config package to application domain folder.

Returns
: absolute path to external fonts directory or an empty string

◆ ext_plugin_dir()

const char * ext_plugin_dir	(		)	

return the absolute path of the folder for external plugins (Android: $HOME/plugins) (VOS/VOS2/VOS3/Titus: empty string)

> This folder is used on Android platform only, which must use a separate folder for plugins. External plugins come along with SDI plugin packages and on Android SDI server is not able to write to lib folder (home_lib_dir()) due to missing write permissions. On platforms without external plugins folder (e.g. VOS), this function returns an empty string.

Returns
: absolute path to external plugin directory or an empty string

◆ extract_tar() [1/2]

int sdi::filesystem::extract_tar	(	const std::string &	*tarfile*,
		const std::string &	*destdir*,
		const char *	*regex* = `0`
	)		

helper function to extract all regular files in uncompressed tar file `tarfile` into directory `destdir`. Caller has the option to pass a regular expression `regex` to match specific files to be considered for extraction. Files, which do not match, are skipped/ignored. The regular expression is applied on the filename of the tar file entry, which looks as follows: Examples: file.json (file on root level) dir/file.json (file in subdirectory) If no regular expression is passed (NULL), all regular files of tar file are extracted.

Parameters

[in]	tarfile	tar file, which shall be extracted
[in]	destdir	destination directory in which all files are extracted. If the directory does not exist yet, it will be created.
[in]	regex	regular expression to match specific files be considered or NULL to extract all regular files from the tar file.

Returns

number of files, which were extracted. -1 is returned in case of error (e.g. tar file is not found, not valid or is empty).

The function does not consider empty directories, in addition, other files types (e.g. symbolic links) are ignored.

◆ extract_tar() [2/2]

int sdi::filesystem::extract_tar	(	const string &	*tarfile*,
		const string &	*destdir*,
		const char *	*regex*
	)		

◆ factory_reset()

void factory_reset	(		)	

perform a factory reset of SDI server. The function is invoked for external reset by command "Factory Reset (20-22)" and for internal reset by init(), if the SDI mode has changed since last startup. The function does the following:

- It removes all writable files, which are modified by SDI server during runtime. In addition, all user config files are removed, which must be considered by the reset. On Titus, even system configuration files are removed.
- It creates writeable files (not belonging to user configuration packages) to restore other default settings, e.g. STATUS.CFG.
- On VOS platforms it synchronizes contents of installed user config packages to restore default configurations files coming along with these packages (see step 5 of init()). On Android user configuration files and SDI plugins are recovered from persitent partition. For

this, functions install_configuration() and install_sdi_plugins() with flag `recover=true` are invoked. On Titus user and system configuration files are recovered from persist folder. For this, functions install_configuration() with flag `recover=true` is invoked.

◆ get_dir_files() [1/2]

int sdi::filesystem::get_dir_files	(	const char *	*dir*,
		std::vector< std::string > *	*files* = `0`,
		const char *	*regex* = `0`
	)		

helper function to count and/or obtain all files in directory `dir`. Caller has the option to pass a regular expression `regex` to match specific files to be considered for the result. The regular expression is applied on the full file path including the path prefix. If no regular expression is passed (NULL), all regular files in the directory are considered. For just counting files of the directory `dir`, caller can set parameter `files` to NULL.

Parameters

[in]	dir	path to directory, which contains the files
[out]	files	pointer to vector storing the found files (absolute file paths) or NULL if the function is just used for file counting.
[in]	regex	regular expression to match specific files be considered or NULL to find all regular files in directory `dir`.

Returns

number of files, which were found. -1 is returned in case of error (e.g. from directory path or missing permissions).

The function does not work recursive, thus, subfolders in directory `dir` are not considered.

◆ get_dir_files() [2/2]

int sdi::filesystem::get_dir_files	(	const char *	*dir*,
		vector< string > *	*files*,
		const char *	*regex*

	)		

◆ home_config_dir()

const char * home_config_dir	(		)	

return the abloslute path of SDI configuration folder in home directory. (VOS/VOS2: $HOME/share/sdi) (VOS3/Android: $HOME/sdi) (Titus: $HOME, with /sdi as $HOME) This folder holds the SDI default configuration files, which are used, as long as not overloaded by external user configuration package. SDI configuration folder contains files for read access only.

Returns
: absolute path to SDI default configuration folder (with read-only files)

◆ home_config_file() [1/2]

std::string sdi::filesystem::home_config_file	(	const std::string &	*file*	)	

Appends a relative filename `file` to path returned by home_config_dir()

Parameters

[in]	file	relative filename to append to home_config_dir()

Returns
: file path

Use this function for SDI configuration file provided by SDI installation packages and which cannot be overloaded by user configuration packages.

◆ home_config_file() [2/2]

string sdi::filesystem::home_config_file	(	const string &	*file*	)	

◆ home_flash_dir()

const char * home_flash_dir	(		)	

return the absolute path to SDI flash folder in home directory. (VOS/VOS2/VOS3/Android: $HOME/flash/sdi) (Titus: $HOME/flash, with /sdi as $HOME) If subfolders do not already exist, they are created with the first call of this function. The flash folder is the location for files, which require write access by SDI server. Writeable configuration files are synchononized with those files of external user configuration packages by invocation of init() function at SDI server startup.

Returns
  absolute path to SDI flash folder for writeable files

◆ home_flash_file() [1/2]

std::string sdi::filesystem::home_flash_file	(	const std::string &	*file*	)	

Appends a relative filename `file` to path returned by home_flash_dir()

Parameters

[in]	file	relative filename to append to home_flash_dir()

Returns
  file path

◆ home_flash_file() [2/2]

string sdi::filesystem::home_flash_file	(	const string &	*file*	)	

◆ home_lib_dir()

const char * home_lib_dir	(		)	

return the abloslute path of lib folder in home directory. (VOS/VOS2/VOS3/Android: $HOME/lib) (Titus: $HOME, with /sdi as $HOME, even libraries are not supported on this platform)

This library folder is used for SDI plugins and libraries coming along with SDI download packages.

Returns

absolute path to home lib folder

◆ init()

void init	(	enum config::SdiSysConfig ::SDIMode	*sdi_mode*	)	

This function must be called at startup, before SDI server accesses other functions of the filesystem module. The function internally does the following:

1. It checks the $HOME environment variable. If it is not provided by the system, the function sets $HOME to working directory of SDI server, since this environment variable is referred the most filesystem functions.
2. VOS/VOS2: It calls vos_wait_for_sdicleaner() to wait/synchronize with sdicleaner installed with a previous removal package.
3. VOS/VOS2/Android: It checks for old files of previous SDI server versions and if exist, these files were taken over to new destination, which is expected by the recent SDI server. Old obsolete files of previous SDI server versions were removed.
4. Create subfolders in home flash folder (home_flash_dir()) so that SDI can create files at startup.
5. Check for validity of configuration files and delete them to recover them with check_config_update() afterwards (see step 8).
6. Read STATUS.CFG (SDI status runtime file). If it does not exist, it is created with defaults. 7 It checks recent SDI mode (passed as parameter *sdi_mode*) equals mode in STATUS.CFG, for which SDI was previously started. If the mode has changed, an internal factory reset (see function factory_reset()) is triggered, since home flash directory with previous files of old mode must be wiped.
7. VOS/VOS2/VOS3: it calls check_config_update() to synchronize files, which were installed with user configuration packages and which require write access. These files are taken over to home flash directory so that SDI is able have write permissions on them. For this, SDI server holds a registration file with a checksum for each file to detect, if it was overloaded by file of the user configuration package. In addition, after removal of a user configuration package, the associated files are removed again and default files from SDI configuration folder are restored. Titus: it calls check_config_update() to checks for installation of user/system configuration files. If installation is pending, the configuration will be processed.
8. It removes files of software upload folder (see upload_install_dir()), if download packages were uploaded by software upload commands (20-14,20-15,20-16) of a previous SDI startup.
   Parameters

[in]	sdi_mode	SDI mode, how SDI was started for this runtime, either SDIMode_Headless or SDIMode_Standard, see enum SDIMode of sdi_sysconfig.h

◆ install_emv_config_package() [1/2]

unsigned short sdi::filesystem::install_emv_config_package	(	const std::string &	*emv_config_pkg*	)	

install an EMV coniguration package as a tarball (uncompressed), which contains EMV configuration files to be installed into EMV flash directory. This is provided by emv_flash_dir() and contains EMV configuration files on which ADKEMV requires write access. The function is called during "SW upload 20-14/20-15/20-16" using upload type UPLOAD_TYPE_EMV_CONFIG_PACKAGE. After all configuration files were installed the function runs required post actions, which are related to the installed files.

Parameters

[in]	emv_config_pkg	file path to EMV config update package (tarball)

Returns

SDI error code (0x9000 for success, 0x64xx for error)

◆ install_emv_config_package() [2/2]

unsigned short sdi::filesystem::install_emv_config_package	(	const string &	*emv_config_pkg*	)	

◆ install_sdi_plugins()

unsigned short install_sdi_plugins	(	bool	*recover* = `false`	)	

install SDI plugins from update package, which was added to SDI update directory (see sdi_update_dir()). Plugins are expected in a specified subfolder "plugin". The function is called wiht command "Check for update (20-1D)" using upload type UPLOAD_TYPE_PLUGIN. On Android the update package data is provided and the command is sent by Android Secure Installer. The function copies the external plugin in internal location, which is specified by ext_plugin_dir(). In addition, the function does not allow to install arbitrary files, therefore, it checks, if each found plugin matches the plugin pattern PLUGIN_PATTERN. After the plugins were overtaken to plugin directory, the plugins are loaded and registered for usage in SDI server. If an external plugin has the same name as an internal plugin (provided with SDI base package, see function home_lib_dir()) and shall "overload" it, the internal plugin is unregisted and unloaded before the external plugin installed. Finally, the function stores a backup copy of each plugin to persitent directory (see sdi_persist_dir()) to make the installed plugins recoverable for command "Factory Reset (20-22)". For command "Factory Reset (20-22)" the same function is invoked with flag `recover=true`, to restore and install the backup copies from persistent directory.

Parameters

[in]	recover	flag set to true to restore and install the SDI plugins from persistent directory

Returns
SDI error code (0x9000 for success, 0x64xx for error)

◆ install_user_configuration()

unsigned short install_user_configuration	(	bool	*recover* = `false`	)	

install user configuration from update package, which was added to SDI update directory (see sdi_update_dir()). The function is called with command "Check for update (20-1D)" using upload type UPLOAD_TYPE_CONFIG_SDIEMV. On Android the update package data is provided and the command is sent by Android Secure Installer. Configuration files are expected in a specified folder structure, whereas other components than SDI (e.g. ADKEMV or ADKSEC) use a path prefix like "emv" or "sec". The function knows the internal location for each file. In addition, the function does not allow to install arbitrary files, therefore, it checks, if the found files are in internal whitelist (see table allowed_files in filesystem.cpp). After all configuration files were installed the function runs required post actions (if any), which are related to the installed files. Finally, the function stores a backup copy of each file to persitent directory (see sdi_persist_dir()) to make the installed files recoverable for command "Factory Reset (20-22)". For command "Factory Reset (20-22)" the same function is invoked with flag `recover=true`, to restore and install the backup copies from persistent directory.

Parameters

[in]	recover	flag set to true to restore and install the configuration files from persistent directory

Returns
SDI error code (0x9000 for success, 0x64xx for error)

◆ log_config_dir()

const char * log_config_dir	(		)	

return the absolute path to folder for ADK Logging configuration files (VOS/VOS2: /mnt/flash/etc/config/adk-log) (Android: $HOME) (VOS/VOS3: /mnt/appdata/versioned/globalshare/etc/config/adk-log) (Titus: <ext_cfg_dir>/adk-log; <ext_cfg_dir> as provided by ext_config_dir())

Required for SW update on Android. On VOS it is unused.

Returns
absolute path to HOME for ADK Logging configuration files

◆ lookup_config_file() [1/2]

std::string sdi::filesystem::lookup_config_file	(	const std::string &	*file*	)	

lookup a configuration file (read-only) by its relative path `file` and return the absolute path for it. The function looks up the file at first in external configuration folder (provided by ext_config_dir()). If not found there, the file is searched in SDI default configuration folder (provided by home_config_dir()). The function return an empty string, if the file is not found in one of both locations.

Parameters

[in]	file	relative file path to look up the configuration file (e.g. "sec/sccfg.json")

Returns
absolute path of the found configuration file (e.g. "/etc/config/sdi/sec/sccfg.json" or "$HOME/share/sdi/sec/sccfg.json") or an empty string, if the file was not found.

◆ lookup_config_file() [2/2]

string sdi::filesystem::lookup_config_file	(	const string &	*file*	)	

◆ move_file() [1/2]

bool sdi::filesystem::move_file	(	const std::string &	*src,*
		const std::string &	*dest*
	)		

helper function to move file from source `src` to destination `dest`.

Parameters

[in]	src	source file location
[in]	dest	destination file location

Returns

true for success, else false for error

◆ move_file() [2/2]

bool sdi::filesystem::move_file	(	const string &	*src*,
		const string &	*dst*
	)		

◆ nfc_flash_dir()

const char * nfc_flash_dir	(		)	

return the absolute path for destination folder of ADKNFC configuration files These files require write access, therefore, on Engage these are located in flash. (VOS/VOS2/VOS3: $HOME/flash) (Android: $HOME) (Titus: $HOME, with /sdi as $HOME, even NFC is not supported on this platform)

Returns

absolute path to HOME for ADKNFC configuration files

◆ plugin_config_dir()

const char * plugin_config_dir	(		)	

return the absolute path of the folder containing plugin configuration files (with extension cfg, json). VOS3/Android: $HOME/sdi/plugincfg VOS/VOS2: $HOME/share/sdi/plugincfg Titus: $HOME/plugincfg, with /sdi as $HOME (unused so far)

Plugin configuration files are installed with SDI plugin package and treaded like internal SDI configuration (as installed with a SDI config package). The reason is these files are installed with system priviledges, thus, we place them to subfolder plugincfg under home configuration directory (see home_config_dir())

Returns

bsolute path of the folder containing plugin configuration files

◆ read_file() [1/2]

bool sdi::filesystem::read_file	(	const char *	*file*,
		std::string &	*data*
	)		

helper function to read a file into a string

Parameters

[in]	file	name of the file to be read
[out]	data	content of the file that was read

Returns

true for success, else false (file couldn't been opened)

◆ read_file() [2/2]

bool sdi::filesystem::read_file	(	const char *	*file*,
		string &	*data*
	)		

◆ remove_dir() [1/2]

int sdi::filesystem::remove_dir	(	const std::string &	*path*,
		unsigned	*modes* = `FM_Default`
	)		

helper function to remove a directory recursively with all its content. If *path* already refers a file, the file is removed.

Parameters

[in]	path	path to drectory (or file) to be removed.

[in]	modes	function operation modes, see enum FilesModes

Returns
number of files, which were removed. -1 is returned in case of error due to missing directory or file access.

◆ remove_dir() [2/2]

int sdi::filesystem::remove_dir	(	const string &	*path*,
		unsigned	*modes*
	)		

◆ remove_files() [1/2]

int sdi::filesystem::remove_files	(	const std::vector< std::string > &	*files*,
		unsigned	*modes* = `FM_Default`
	)		

helper function to remove multiple files. Files to be removed are specified by vector `files`.

Parameters

[in]	files	vector of files to be removed
[in]	modes	function operation modes, see enum FilesModes

Returns
number of files, which were removed. -1 is returned in case of error due to missing directory or file access.

◆ remove_files() [2/2]

int sdi::filesystem::remove_files	(	const vector< string > &	*files*,
		unsigned	*modes*
	)		

◆ remove_sdi_plugins()

unsigned short remove_sdi_plugins	(		)	

remove SDI plugins according removal file, which is looked up in SDI update directory (see sdi_update_dir()). The removal file is expected in a specified subfolder "plugin". The function is called with command "Check for update (20-1D)" using upload type UPLOAD_TYPE_REMOVE_PLUGIN. On Android the update package data is provided and the command is sent by Android Secure Installer. Before removal of a plugin, the plugin is unloaded and unregistered for usage in SDI server. If a plugin is removed, which has the same name as an internal plugin in home_lib_dir(), the internal plugin is loaded and reactivated again. Finally, the function removes the backup copy of each plugin from persitent directory (see sdi_persist_dir()) so that command "Factory Reset (20-22)" will no longer recover the plugin.

The removal file (remove.json) contains an array 'files' with a filename per plugin to remove. No path prefix is added, the function knows the internal location for each file. The filenames may also contain reguluar expressions to match multiple files to delete. The function does not allow to remove arbitrary files, therefore, it checks, if the regular expression matches one or more plugins is in external plugin folder before the plugin is unregistered, unloaded and removed. If a filename in the removal file does not match any existing plugin, the entry is ignored and it is proceeded with the next one. Example format for removal.json:

```
{ "files": [ "libsdiplugin.*\.so", // remove all external SDI plugins ]}
```

Returns
SDI error code (0x9000 for success, 0x64xx for error)

◆ remove_user_configuration()

unsigned short remove_user_configuration	(		)	

remove user configuration files according removal file, which is looked up in SDI update directory (see sdi_update_dir()). The function is called with command "Check for update (20-1D)" using upload type UPLOAD_TYPE_REMOVE_CONFIG_SDIEMV. On Android the update package data is provided and the command is sent by Android Secure Installer. After removal, it runs required post actions (if any), which are related to the removed files. Finally, the function removes the backup copy of each file from persitent directory (see sdi_persist_dir()) so that command "Factory Reset (20-22)" will no longer recover the files.

The removal file (remove.json) contains an array 'files' with a relative path per file to remove. The same external path representation as in update packages is used and the function knows the internal location for each file. The lines may also contain reguluar expressions to match multiple files to delete. The function does not allow to remove arbitrary files, therefore, it checks, if the found files are in internal whitelist (see table allowed_files). If a line in the removal file does not match any existing allowed file, the line is ignored and it is proceeded with the next line. Example format for removal.json:

```
{ "files": [ "emv/[Ee][Mm][Vv][_-].*\.[Xx][Mm][Ll]", // remove all EMV config files
"sec/sccfg.json", "config.json", "whitelist.json", "sensitivetags.json", "cardranges.json" ]}
```

Returns

SDI error code (0x9000 for success, 0x64xx for error)

◆ sdi_certstore_dir()

const char * sdi_certstore_dir	(		)	

certificate directory, in which SDI stores certificates (e.g. CAs for TLS (Authex) and device pairing). Usually, the function is required for devices with K81, from which certificates are read before used. (Android: $HOME/certstore) (VOS/VOS2/VOS3: $HOME/flash/sdi/certstore)

Returns

ablosulte path to SDI certstore folder

◆ sdi_persist_dir()

const char * sdi_persist_dir	(	bool	*system* = `false`	)	

persistant directory to store SDI related update files for default recovery. (Android: /persist/appdata/sdi) (VOS/VOS2/VOS3: empty string) (Titus: $HOME/persist/sys, for system==true $HOME/persist/usr, for system==false) Recently used the following platforms: Android: command "Check for update (20-1D)" and command "Factory Reset (20-22)" Titus: installation for user/system configuration package installation with command "Software Upload (20-14,20-15,20-16)" and command "Factory Reset (20-22)"

Returns

ablosulte path to persistent SDI data folder or an empty string, if not supported on platform (e.g. Engage)

◆ sdi_update_dir()

const char * sdi_update_dir	(		)	

read-only source directory for SDI related update package files. (Android: /data/secure/sdi) (VOS/VOS2/VOS3: empty string) (Titus: $HOME/update) Recently used the following platforms: Android: command "Check for update (20-1D)" Titus: installation for user/system configuration package installation with command "Software Upload (20-14,20-15,20-16)"

Returns
    ablosulte path to update package folder or an empty string, if not supported by plattform (e.g. Engage)

◆ sys_remove_sponsor_dir()

const char * sys_remove_sponsor_dir	(		)	

return absolute path to system directory for removal of sponsor certificates (VOS/VOS2/VOS3/Titus: not supported, an empty string is returned) (Android: /data/secure/sponsor -> used to store the file CRTRESET.SYS to remove the sponsor certificate).

> This is recently required by SDI server for command 20-1D (Check for Update) with command type 7 (remove sponsor certificate).

Returns
    absolute path to system sponsor removal directory (or an empty string if not supported)

◆ tmp_dir()

const char * tmp_dir	(		)	

return the absolute path to writeable system temp folder. (VOS/VOS2/VOS3: /tmp) (Android: $HOME/tmp) (Titus: $HOME/tmp, with /sdi as $HOME, unused so far)

> This is recently required by SDI server to store some temporary keys for VCL.

Returns
    absolute path to writeable system temp folder

◆ upload_install_dir()

const char * upload_install_dir	(	bool	*flash* = `true`	)	

return absolute path to upload and installation directory for download packages or certificates (VOS/VOS2: /mnt/flash/install/dl or $HOME/flash/sdi/install) (VOS3: $HOME/flash/sdi/install) (Android: $HOME/install -> e.g. used to install the sponsor certificate) (Titus: $HOME/install, with /sdi as $HOME)

This is recently required by SDI server to store download packages with software upload commands (20-14,20-15,20-16) and sponsor certificate installation with command 20-1B.

Parameters

[in]	flash	set to true to use home flash folder as installation directory. This parameter is relevant for VOS/VOS2 only, which requires to use home flash directory to store temporary installation files for newer OS versions. This is to reduce RAM usage, since files would be usually stored to /mnt/flash/install/dl, which is a RAM disk.

Returns
absolute path to system download package installation directory

◆ write_file() [1/2]

bool sdi::filesystem::write_file	(	const char *	*file*,
		const std::string &	*data*
	)		

helper function to write a string into a file

Parameters

[in]	file	name of the file to be written

[out]	data	content of the string to be written

Returns

true for success, else false (file couldn't been opened)

◆ write_file() [2/2]

bool sdi::filesystem::write_file	(	const char *	*file*,
		const string &	*data*
	)		

Variable Documentation

◆ allowed_usr_files

```
const struct UpdateFiles allowed_usr_files[]
```

Initial value:

```
= { { EMV_PREFIX, EMV_FLASH_CONFIG_FILES, true, LOC_EmvFlashDir, "", (ACT_DisableEpp | ACT_EmvExit),
ACT_EmvInitInfo }, { EMV_PREFIX, EMV_DESIRED_FILE, false, LOC_EmvConfigDir, "", (ACT_DisableEpp |
ACT_EmvExitComplete), ACT_EmvInitInfo }, { SEC_PREFIX, ADKSEC_CONFIG_FILE, false, LOC_SdiExtConfigDir,
SEC_PREFIX, ACT_None, ACT_SecInit }, { "", SDI_CONFIG_FILE, false, LOC_SdiExtConfigDir, "", ACT_None, ACT_None
}, { "", SDI_ACL_FILE, false, LOC_SdiExtConfigDir, "", ACT_None, ACT_AclInit }, { "", WHITELIST_FILE, false,
LOC_SdiFlashDir, "", ACT_None, ACT_WhitelistFile }, { "", SENSITIVE_TAGS_FILE, false, LOC_SdiFlashDir, "",
ACT_None, ACT_None }, { "", CARD_RANGES_FILE, false, LOC_SdiFlashDir, "", ACT_None, ACT_CardRanges }, { "",
UPDATE_REMOVE_FILE, false, LOC_None, "", ACT_None, ACT_None }, { LOG_PREFIX, LOG_CONFIG_FILES, true,
LOC_LogConfigDir, "", ACT_None, ACT_LogInit }, { NFC_PREFIX, NFC_WKY_FILES, true, LOC_NfcFlashDir, "", ACT_None,
ACT_None }, { FONT_PREFIX, FONT_FILES, true, LOC_SdiExtFontsDir, "", ACT_None, ACT_None } }
```

user configuration files allowed for update and removal

◆ allowed_usr_files_size

```
const unsigned allowed_usr_files_size = (sizeof(allowed_usr_files) / sizeof(allowed_usr_files[0]))
```