Tizen Content Screening

API Specification

Document version 1.1.3

Document Information

Document Details

Revision	1.1.3
Author	MMS Development Team

Revision Information

Revision	**Revision Date**	**Author**	**Details**
1.0.0	09/05/2012	MMS Development Team	Created
1.0.1	10/05/2012	MMS Development Team	Add implementation guide Add data type javascript Add compression flag Change file system module to application launcher
1.0.2	11/05/2012	MMS Development Team	Remove scan open API.
1.0.3	01/26/2013	MMS Development Team	Add license.
1.1.0	03/28/2014	MMS Development Team	Add details for TCSGetVersion API.
1.1.1	06/24/2014	MMS Development Team	Updated for scan cancel feature and TCSGetInfo API. Updated document for Async scan functions.
1.1.2	07/09/2014	MMS Development Team	Added TCSOffset and TCSErrorCode. Fix TCSScanParam info.
1.1.3	07/30/2014	MMS Development Team	Backward compatibility fix for scan API.

Contents

Terms, Abbreviations, Definitions, Conventions

Items	Description
SDK	Software Development Kit
API	Application Programming Interface
Content Screening	Screening content for security consideration
Module	Program, service or any execution entity in the Tizen platform
Application	Executable provided by either system or third-party

Overview

This document defines the Content Screening API for Tizen platform. The API enables caller modules and applications to scan the content inside their logic data. The Content Screening API is defined in native C language. A computer language bundle might be required if calling from any other language. For example, if we want to call Content Screening API from Java, we need to add JNI code (language bundle) to enable Java code to call Content Screening API from virtual machine.

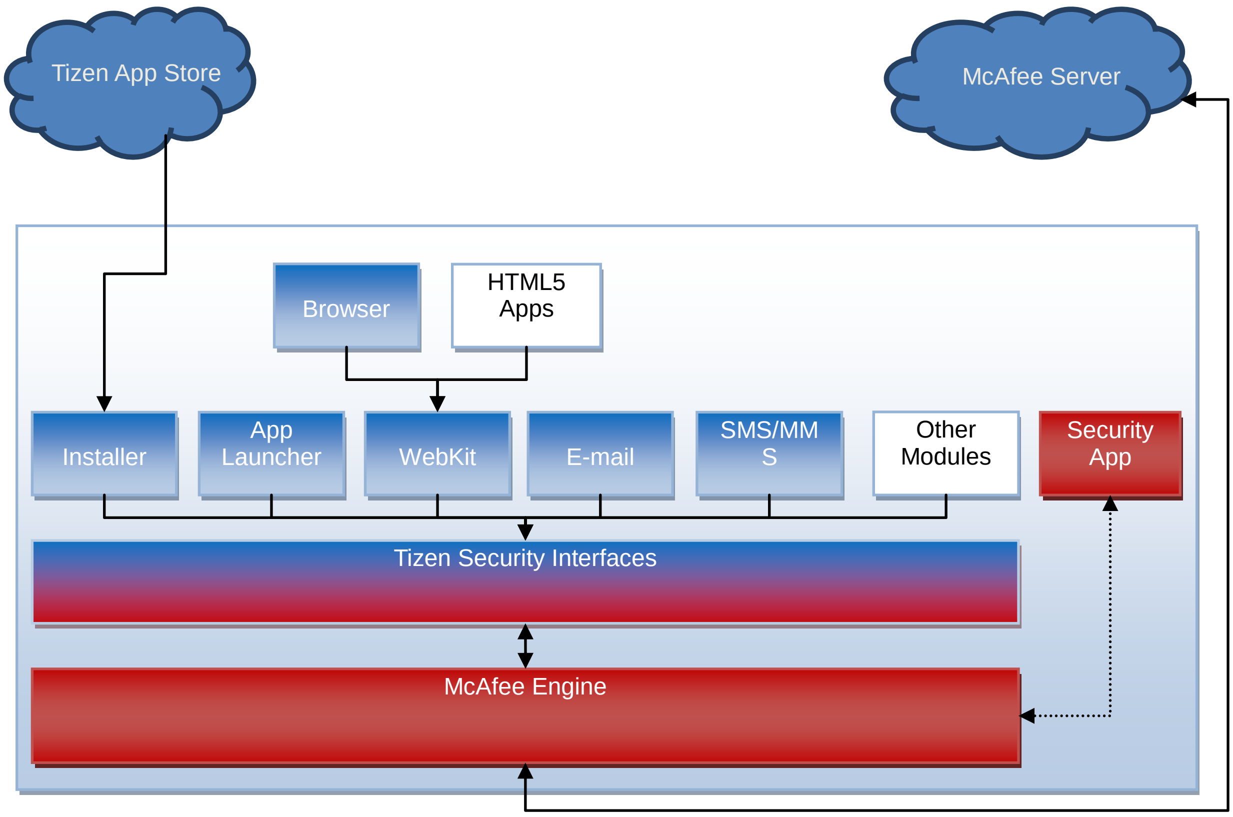


Figure 1 Overview of Content Screening

This document is to define the API specification in Tizen Security Interfaces. As for McAfee Engine API please reference to McAfee MCS API specification, which is out of the scope of this document.

The API is composed by following categories:

- Initialization and clean-up functions
- Scan functions
- Support functions

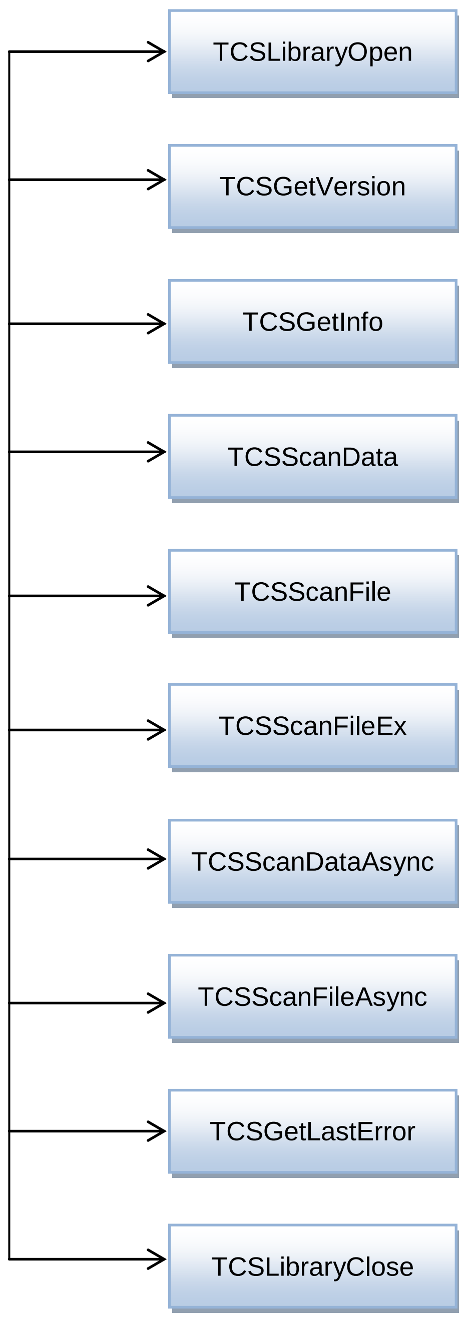


Figure 2 API Call Diagram

`TCSLibraryOpen()` and `TCSLibraryClose()` are used to initialize Tizen Content Screening library or clean up resource on application exit. `TCSGetLastError()` is a support function to return caller an error code to indicate the latest error during this library function call.

`TCSScanData()`and `TCSScanDataAsync()` are used to scan content in the memory while `TCSScanFileEx()`and `TCSScanFileAsync()` are used to scan the content on permanent storages, like SD card.

`TCSGetInfo()` and `TCSGetVersion()` are used to get Meta Information and Version of framework/plugin respectively.

Library

The delivery of Tizen Content Screening API should be a .so library: `libsecfw.so`

Initialize Functions

Summary

Methods	
`TCSLIB_HANDLE`	`TCSLibraryOpen()`
	Initialize Tizen Content Screening library.
`void`	`TCSLibraryClose(TCSLIB_HANDLE hLib)`
	Close Tizen Content Screening library by releasing all resources it occupied.

Methods

```
TCSLIB_HANDLE TCSLibraryOpen()
```

Initialize Tizen Content Screening library. For example, allocating memory for internal data use, loading signature database, etc.

Parameters

None.

Returns

An instance of Tizen Content Screening library context – on success.
INVALID_TCSLIB_HANDLE – on failure.

```
void TCSLibraryClose(TCSLIB_HANDLE hLib)
```

Destroy Tizen Content Screening library instance. Release all resources it occupies.

Parameters

`hLib`	Tizen Content Screening library instance returned by `TCSLibraryOpen()`.

Returns

None.

Get Version Function

Summary

Methods	
int	`TCSGetVersion(TCSLIB_HANDLE hLib, TCSVerInfo *pVerInfo);`
	Returns the version number of Framework and Plugin.

Methods

```
int TCSGetVersion(TCSLIB_HANDLE hLib, TCSVerInfo *pVerInfo)
```

Returns the version number of Framework and Plugin.

Parameters

`hLib`	Tizen Content Screening library instance returned by `TCSLibraryOpen()`.
`pVerInfo`	Pointer to a structure `TCSVerInfo` to get version information of framework and plugin.

Returns

`0` – on success.
`-1` – on failure

Get Info Function

Summary

Methods	
int	`TCSGetInfo(TCSLIB_HANDLE hLib, char *pszInfo);`
	Returns the Meta information about Plugin.

Methods

```
int TCSGetInfo(TCSLIB_HANDLE hLib, char *pszInfo);
```

Returns the Meta information about Plugin.

Parameters

`hLib`	Tizen Content Screening library instance returned by `TCSLibraryOpen()`.
`pszInfo`	String containing the meta information

Returns

`0` – on success.
`-1` – on failure

Scan Functions

Summary

Methods	
int	`TCSScanData(TCSLIB_HANDLE hLib, TCSScanParam *pParam, TCSScanResult *pResult)`
	Scan content in memory.
int	`TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName, int iDataType, int iAction, int iCompressFlag, void *pPrivate, int (*pfCallBack)(void *pPrivate, int iReason, void *pParam), TCSScanResult *pResult)`
	Scan content in file system.
int	`TCSScanDataAsync(TCSLIB_HANDLE hLib, TCSScanParam *pParam);`
	Scan content in memory asynchronously.
int	`TCSScanFile(TCSLIB_HANDLE hLib, char const *pszFileName, int iDataType, int iAction, int iCompressFlag, TCSScanResult *pResult);`
	Legacy API for scanning content in file system synchronously.
int	`TCSScanFileEx(TCSLIB_HANDLE hLib, char const *pszFileName, int iDataType, int iAction, int iCompressFlag, void *pPrivate, int (*pfCallBack)(void *pPrivate, int iReason, void *pParam), TCSScanResult *pResult);`
	Scan content in file system synchronously.
int	`TCSScanFileAsync(TCSLIB_HANDLE hLib, char const *pszFileName, int iDataType, int iAction, int iCompressFlag, void *pPrivate, int (*pfCallBack)(void *pPrivate, int iReason, void *pParam));`

	Scan content in file system asynchronously.

Methods

```
int TCSScanData (TCSLIB_HANDLE hLib,
                 TCSScanParam *pParam,
                 TCSScanResult *pResult)
```

Scan content in memory. Caller need to pass callback functions in `pParam` so that scanner can read or write data back and forth. Scan result will be returned in a data structure `pResult`. The integer return value of this function call is just to indicate if this call is success or not. For any failure of this function call please use `TCSGetLastError()` to get detail information.

Parameters

`hLib`	Tizen Content Screening library instance returned by `TCSLibraryOpen()`.
`pParam`	Memory address of data structure instance `TCSScanParam`, see detail at TCSScanParam.
`pResult`	Memory address of data structure instance `TCSScanResult`, see detail at TCSScanResult.

Returns

`0` – on success.
`-1` – on failure

```
int TCSScanFile (TCSLIB_HANDLE hLib,
                 char const *pszFileName,
                 int iDataType,
                 int iAction,
                 int iCompressFlag,
                 TCSScanResult *pResult)
```

Legacy API for scanning content on file system, keep it for backward compatible. It requires scanner instance, file path, data type the file could be, and type of action for malware. It will return detail scan result in data structure instance `pResult`. The integer return value of this function call indicates if the call is success or not. For any failure of this function call please use `TCSGetLastError()` to get detail information.

Parameters

`hLib`	Tizen Content Screening library instance returned by `TCSLibraryOpen()`.
`pszFileName`	Path to the file.
`iDataType`	Data type of the file. It could be set to unknown type, which leaves the scanner to determine. But by specifying the data type, potentially can accelerate the scanning progress. For detail information please see Data Type.
`iAction`	Action type if malware detected. Please find detail at Action Type.
`iCompressFlag`	`0` – decompression disabled, `1` – decompress enabled
`pResult`	Memory address of data structure instance `TCSScanResult`, see detail at TCSScanResult.

Returns

0 – on success.
-1 – on failure

```
int TCSScanFileEx (TCSLIB_HANDLE hLib,
                   char const *pszFileName,
                   int iDataType,
                   int iAction,
                   int iCompressFlag,
                   void *pPrivate,
                   int (*pfCallBack)(void *pPrivate,
                                     int iReason,
                                     void *pParam),
                   TCSScanResult *pResult)
```

Scan content on file system. It requires scanner instance, file path, data type the file could be, and type of action for malware. It will return detail scan result in data structure instance `pResult`. The integer return value of this function call indicates if the call is success or not. For any failure of this function call please use `TCSGetLastError()` to get detail information.

Parameters

`hLib`	Tizen Content Screening library instance returned by `TCSLibraryOpen()`.
`pszFileName`	Path to the file.
`iDataType`	Data type of the file. It could be set to unknown type, which leaves the scanner to determine. But by specifying the data type, potentially can accelerate the scanning progress. For detail information please see Data Type.
`iAction`	Action type if malware detected. Please find detail at Action Type.
`iCompressFlag`	`0` – decompression disabled, `1` – decompress enabled
`pPrivate`	Caller context data. Instead performing direct access to this field, scanner will pass this context data back to caller via below callbacks, so that caller can track the access status inside their own context data without creating global variables.
`pfCallBack`	It is used by scanner to notify caller for specific events via this callback function. Return 0 to proceed scan and -1 to cancel the scan.
`pResult`	Memory address of data structure instance `TCSScanResult`, see detail at TCSScanResult.

Returns

0 – on success.
-1 – on failure

```
int TCSScanDataAsync (TCSLIB_HANDLE hLib, TCSScanParam *pParam)
```

Asynchronously scan contents in memory. Caller needs to pass callback functions in `pParam` so that scanner can read or write data back and forth. The method returns after starting a new thread to run scan. When the scan finishes, the callback is called with the scan result structure `TCSScanResult`.

Parameters

`hLib`	Tizen Content Screening library instance returned by `TCSLibraryOpen()`.
`pParam`	Memory address of data structure instance `TCSScanParam`.

Returns

`0` – on success.
`-1` – on failure.

```
int TCSScanFileAsync (TCSLIB_HANDLE hLib,
                      char const *pszFileName,
                      int iDataType,
                      int iAction,
                      int iCompressFlag,
                      void *pPrivate,
                      int  (*pfCallBack)(void *pPrivate,
                                         int iReason,
                                         void *pParam));
```

Asynchronously scans content of a file. It requires scanner instance, file path, data type the file could be, and type of action for malware. It will return detail scan result in data structure instance `TCSScanResult`. The integer return value of this function call indicates if thread creation succeeded or not. When the scan finishes, the callback is called with the scan result structure `TCSScanResult`.

Parameters

`hLib`	Tizen Content Screening library instance returned by `TCSLibraryOpen()`.
`pszFileName`	Path to the file to be scanned.
`iDataType`	Data type of the file. It could be set to unknown type, which leaves the scanner to determine. But by specifying the data type, potentially can accelerate the scanning progress. For detail information please see Data Type.
`iAction`	Action type if malware detected. Please find detail at Action Type.
`iCompressFlag`	0 – decompression disabled, 1 – decompression enabled
`pPrivate`	Caller application context data.
`pfCallBack`	Callback method to be notified after the scan.

Returns

`0` – on success.
`-1` – on failure.

Note: The `TCSLibraryClose()` should not be called in the caller's context until the async scan finishes and callback from asynchronous API returns back.

CallBack Functions

Summary

Callback Methods	
`int`	`(*pfCallBack)(void *pPrivate, int iReason, void *pParam)`
	Asynchronous Data Scan.

Methods

```
int (*pfCallBack)(void *pPrivate, int iReason, void *pParam);
```

The above callback method is used to get notification when the data scan finishes. The method runs in the same thread context in which the scan was performed. The scanning thread ends as soon as the callback returns. For the failure reason of the scan operation, use `TCSGetLastError()` to get detail information. Return `0` to proceed scan and `-1` to cancel the scan.

Parameters

`pPrivate`	Caller context data.
`iReason`	Reason for scanner to call caller. Set to TCS_CB_SCANREPORT when asynchronous scan finishes. Please find detail at CallBack Reason.
`pParam`	Pointer to TCSScanResult struct or `NULL` if failure. This is the same structure passed to the Scan API.

Returns

`0` – to proceed scan.
`-1` – to cancel scan.

Support Function

Summary

Methods	
`int`	`TCSGetLastError(TCSLIB_HANDLE hLib);`
	Return last error code.

Methods

```
int TCSGetLastError (TCSLIB_HANDLE hLib)
```

This function is used to retrieve the error code previous function call occurs. All scan functions return zero to indicate success, and `-1` for failure. The error code gives the detail of the failure reason for trouble shooting.

Parameters

`hLib`	Tizen Content Screening library instance returned by `TCSLibraryOpen()`.

Returns

Error code, please find detail at Error Code.

TCSScanParam

Description

Data structure for caller to pass input data for scanning.

Summary

Fields	
`iAction`	The type of action that caller want to take when malware detected. Please find detail at Action Type.
`iDataType`	The type of content data. For example, archived file. Please find detail at Data Type.
`iCompressFlag`	`0` – decompression disabled, `1` – decompression enabled.
`pPrivate`	Caller context data. Instead performing direct access to this field, scanner will pass this context data back to caller via below callbacks, so that caller can track the access status inside their own context data without creating global variables.
`TCSOffset`	`(*pfGetSize)(void *pPrivate)`
	It is used by scanner to obtain content data size in bytes from caller via this callback function.
`int`	`(*pfSetSize)(void *pPrivate, TCSOffset uSize)`
	It is used by scanner to change content data size in bytes via this callback function. (For example, repair infected data)
`unsigned int`	`(*pfRead)(void *pPrivate, TCSOffset uSize, void *pBuffer, unsigned int uCount)`
	It is used by scanner to read content data in bytes from caller via this callback function.
`unsigned int`	`(*pfGetWrite)(void *pPrivate, TCSOffset uSize, void const *pBuffer, unsigned int uCount)`
	It is used by scanner to change content data in bytes via this callback function.
`int`	`(*pfCallBack)(void *pPrivate, int iReason, void *pParam)`
	It is used by scanner to notify caller for specific events via this callback function.

CallBack methods

```
TCSOffset (*pfGetSize) (void *pPrivate)
```

To scan content data in memory, scanner needs to know the size of the data to be scanned. This callback function is supposed to return the content data size in bytes to scanner.

Parameters

`pPrivate`	Caller context data.

Returns

Size in bytes.

```
int (*pfSetSize) (void *pPrivate,
                  TCSOffset uSize)
```

When scanner try to repair destroyed content data by malware, it usually needs to change the size of content data size, so that caller can do coordinate work for this change.

Parameters

`pPrivate`	Caller context data.
`uSize`	New size in bytes

Returns

Size in bytes, not equal to expected size indicating failure of this call.

```
unsigned int (*pfRead) (void *pPrivate,
                        TCSOffset uOffset,
                        void const *pBuffer,
                        unsigned int uCount)
```

When scanner scan the content data in memory it needs to read data from caller instead of directly access the memory, this enables flexibility for scanner to handle variable format of content data and stream scanning.

Parameters

`pPrivate`	Caller context data.
`uOffset`	Offset where to start reading
`pBuffer`	The memory address of buffer which is to be filled with read data as result of this read call.
`uCount`	Bytes to be read

Returns

Read bytes count, not equal to expected size indicating failure of this call.

```
unsigned int (*pfWrite) (void *pPrivate,
                         TCSOffset uOffset,
                         void const *pBuffer,
                         unsigned int uCount)
```

When scanner repairs the broken content data in memory it needs to write data through this callback.

Parameters

`pPrivate`	Caller context data.
`uOffset`	Offset where to start writing
`pBuffer`	The memory address of buffer which is to be copied

	to the specified offset.
`uCount`	Bytes to be written

Returns

Written bytes count, not equal to expected size indicating failure of this call.

```
int (*pfCallBack) (void *pPrivate,
                   int iReason,
                   void *pParam)
```

When scanner repairs the broken content data in memory it needs to write data through this callback.

Parameters

`pPrivate`	Caller context data.
`iReason`	Reason for scanner to call caller. Please find detail at CallBack Reason.
`pParam`	Coordinate parameter for specific reason. Please find detail at CallBack Reason.

Returns

`0` – indicating success
Negative value – indicating stop/cancel scanning

CallBack methods

```
void (*pfFreeResult) (struct TCSScanResult_struct *pResult)
```

Caller has to pass scan result instance back to this callback function to release resources used by detection list.

Parameters

`pResult`	The scan result instance.

Returns

None.

TCSErrorCode

Description

Typedef for error code type.

TCSOffset

Description

Typedef to Support 64 bits data / file locating.

TCSScanResult

Description

Data structure for scanner to pass detail scan result back to caller.

Summary

Fields	
`iNumDetected`	The number of detections.
`pDList`	Detection list, please find detail at TCSDetected.
`void`	`(*pfFreeResult)(struct TCSScanResult_struct *pResult)`
	It is used by caller to release detection list resources when needed.

CallBack methods

```
void (*pfFreeResult) (struct TCSScanResult_struct *pResult)
```

Caller has to pass scan result instance back to this callback function to release resources used by detection list.

Parameters

`pResult`	The scan result instance.

Returns

None.

TCSDetected

Description

Data structure for scanner to pass detection information to caller.

Summary

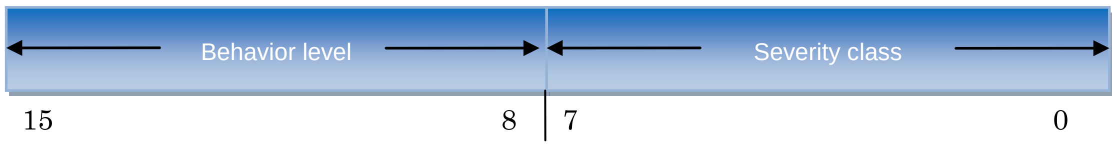
Fields	
`pNext`	Detection structure is a link list node.
`pszName`	Malware name.
`pszVariant`	Malware variant name.
`uType`	Malware type. Please see below table for detail.
`uAction`	Bit-field of malware severity, class and behaviour level. Please find detail in below table.
`pszFileName`	Path of the infected file, it can be ignored if current is a memory data scan.

Malware type

Fields	
`TCS_VTYPE_MALWARE`	It is a malware.
`TCS_VTYPE_PUP`	It is a potentially unwanted program.

Malware action

This is a bit-field variable which contains malware severity flags and application behavior levels in bits. Bits `31` – `16` are reserved.



Behavior level

Fields	
`TCS_BC_LEVEL0`	Process with a warning. The severity is assigned to data previously considered malicious.

TCS_BC_LEVEL1	Prompt user before processing. Ask user if they want the application to process the data.
TCS_BC_LEVEL2	Do not process the data.
TCS_BC_LEVEL3	Do not process the data and prompt user for removal. If the content stored on the terminal, prompt the user for permission before removal.
TCS_BC_LEVEL4	Do not process the data and automatically remove if stored.

Severity class

Fields	
TCS_SC_USER	The malware is harmful to end user.
TCS_SC_TERMINAL	The malware is harmful to the terminal.

TCSVerInfo

Description

Data structure for caller to get version information of framework and plugin.

Summary

Fields	
szFrameworkVer	Framework version.
szPluginVer	Plugin version.

Data Type

Description

Data type that Tizen Content Screening library support.

Summary

Fields	
TCS_DTYPE_UNKNOWN	Data type is unknown, scanner is to determine the data type by itself.
TCS_DTYPE_HTML	HTML content.
TCS_DTYPE_URL	Content data is URL.
TCS_DTYPE_EMAIL	Content data is e-mail.
TCS_DTYPE_PHONE	Content data is phone number.
TCS_DTYPE_JAVA	Content data is Java code.
TCS_DTYPE_JAVAS	Content data is JavaScript.

Fields	
`TCS_DTYPE_UNKNOWN`	Data type is unknown, scanner is to determine the data type by itself.
`TCS_DTYPE_HTML`	HTML content.
`TCS_DTYPE_URL`	Content data is URL.
`TCS_DTYPE_EMAIL`	Content data is e-mail.
`TCS_DTYPE_PHONE`	Content data is phone number.
`TCS_DTYPE_JAVA`	Content data is Java code.
`TCS_DTYPE_JAVAS`	Content data is JavaScript.
`TCS_DTYPE_TEXT`	Content data is plain text.

Action Type

Description

Action type that caller want to take on detected malware.

Summary

Fields	
`TCS_SA_SCANONLY`	Tell scanner scan content data without changing anything.
`TCS_SA_SCANREPAIR`	Tell scanner to repair content data if detected.

CallBack Reason

Description

Reason codes for scanner to callback on caller.

Summary

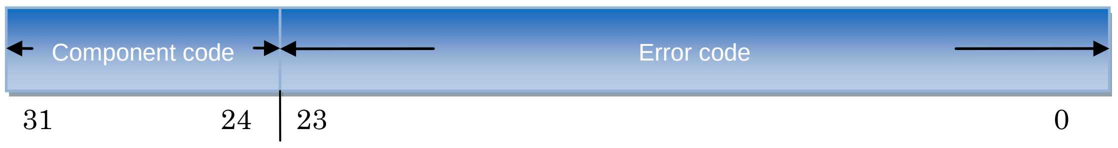
Fields	
`TCS_CB_SCANREPORT`	Tell caller that asynchronous scan has completed.
`TCS_CB_DETECTED`	Tell caller that malware was detected.

The coordinate parameter for this callback reason is TCSDetected

Error Code

Description

Error code definition is a bit-field.



Component code

Fields	
TCS_ERROR_MODULE_GENERIC	Generic error can be occurred in all components.

Error code

Fields	
TCS_ERROR_NOT_IMPLEMENTED	Tizen Content Screening library is not implemented.

Call Sequence examples

Sync call sequence

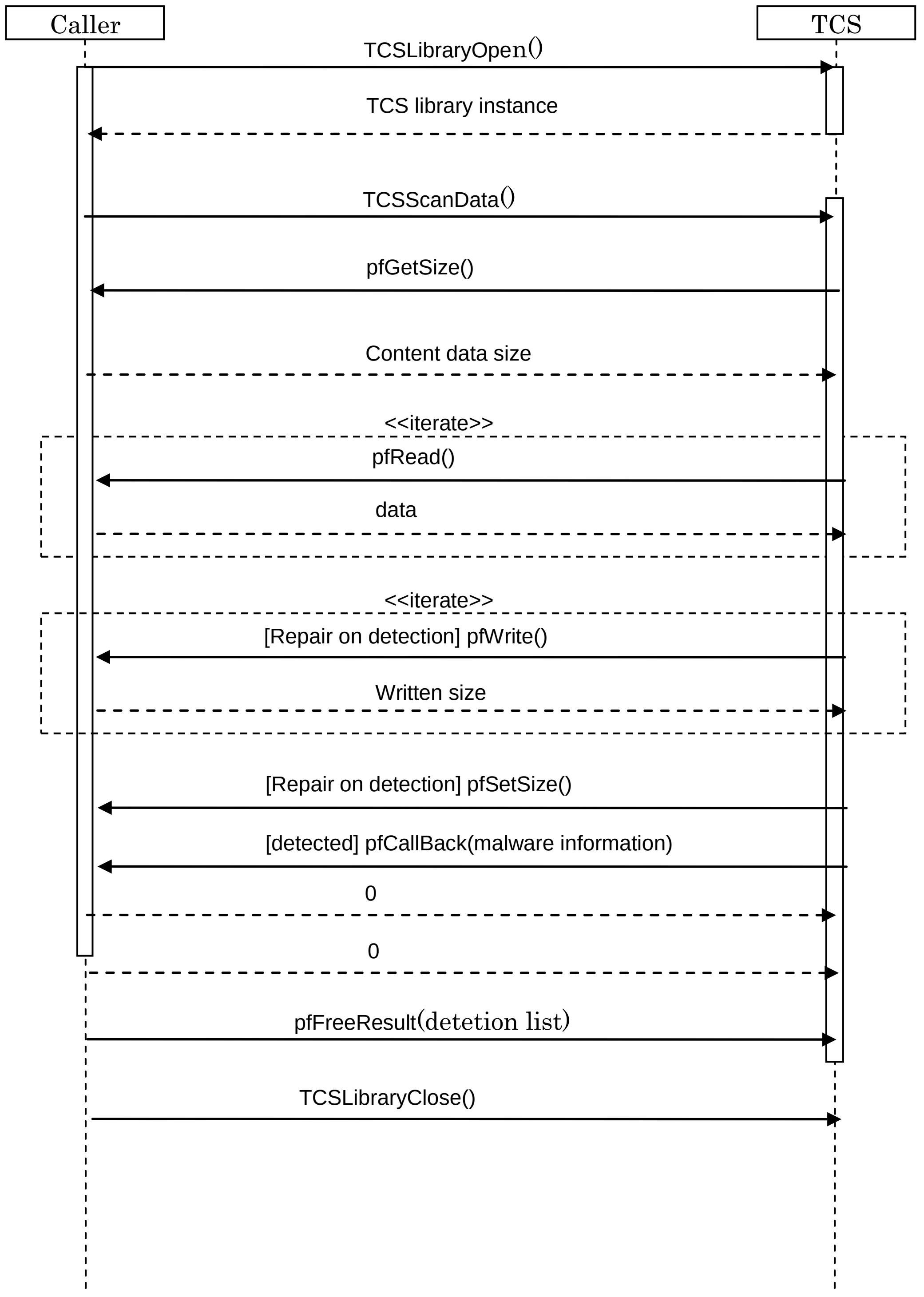


Figure 3

Async call sequence

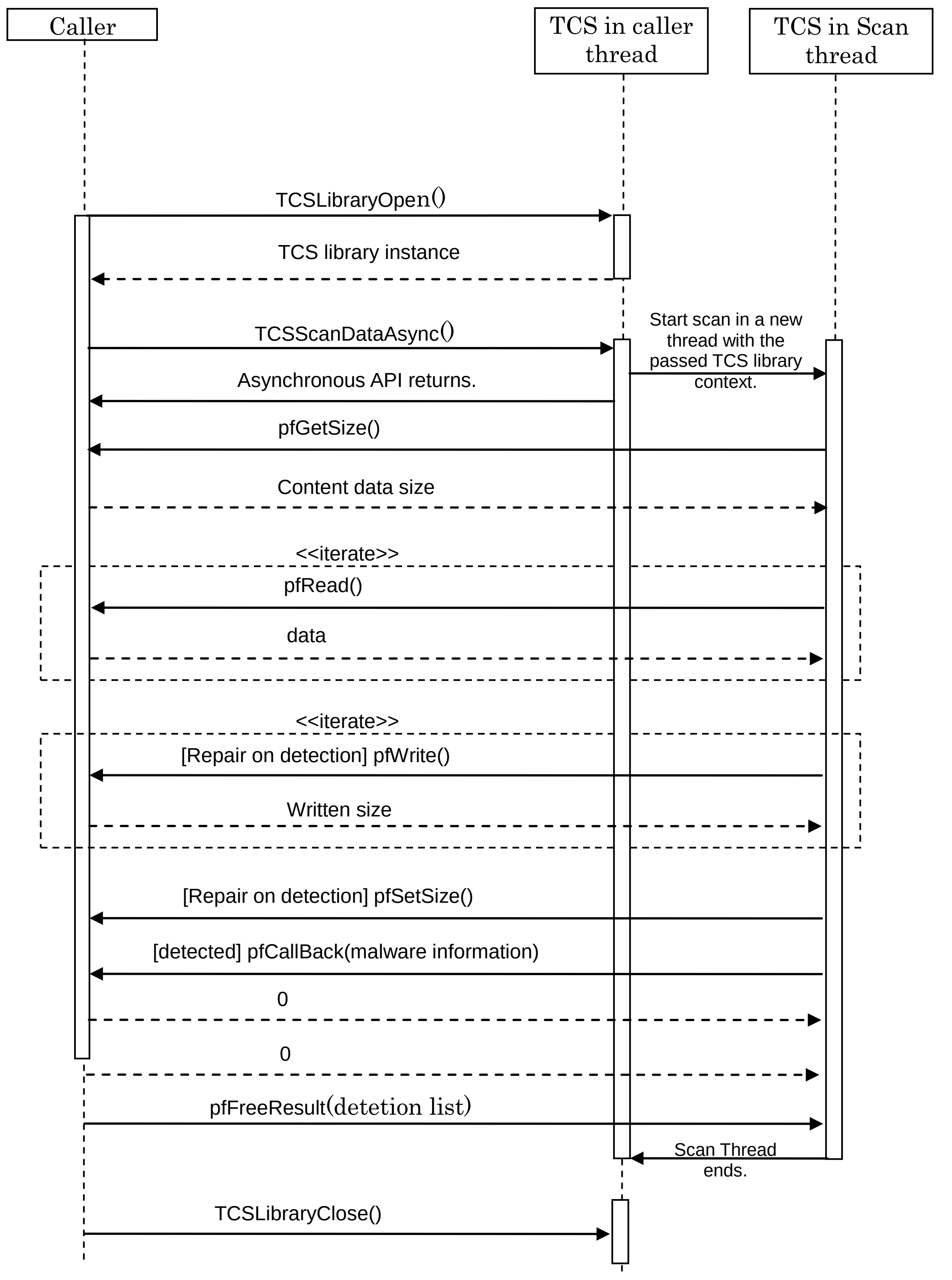

Caller
TCS in caller thread
TCS in Scan thread
TCSLibraryOpen()
TCS library instance
TCSScanDataAsync()
Start scan in a new thread with the passed TCS library context.
Asynchronous API returns.
pfGetSize()
Content data size
<<iterate>>
pfRead()
data
<<iterate>>
[Repair on detection] pfWrite()
Written size
[Repair on detection] pfSetSize()
[detected] pfCallBack(malware information)
0
0
pfFreeResult(detetion list)
Scan Thread ends.
TCSLibraryClose()

Figure 4

Code sample

```
//Sync Scan File
ScanFile(char *pszFilePath)
{
    TCSScanResult SR;
    TCSLIB_HANDLE hLib;
    hLib = TCSLibraryOpen();
    if (hLib == INVALID_TCSLIB_HANDLE)
        return -1;

    TCSScanFile(hLib, pszFilePath,
                TCS_DTYPE_UNKNOWN, TCS_SA_SCANONLY, 1,
                NULL, NULL, &SR);

    if (SR.pfFreeResult != NULL)
        SR.pfFreeResult(&SR);
    TCSLibraryClose(hLib);
}

//Async Scan File
int OnTaskInProgressN(void* pPrivate, int iReason, void* pParam)
{
    switch (iReason)
    {
        case TCS_ASYNCSCAN_REPORT:
        {
          if (pParam != NULL)
          {
                    TCSScanResult *Result = (TCSScanResult*)pParam;
                    if (Result->iNumDetected > 0)
                        printf("Malware found : File Name = %s, Malware Name = %s"
                               ,Result->pDList->pszFileName, Result->pDList->pszName);

                    if (Result->pfFreeResult != NULL)
                        Result->pfFreeResult(&SR);
          }
        }
        break;

    }
  return 0;  //return -1 to cancel the scan
}

int ScanFileAsync(char *pszFilePath)
{
    TCSLIB_HANDLE hLib;
    hLib = TCSLibraryOpen();
    if (hLib == INVALID_TCSLIB_HANDLE)
          return -1;
    TCSScanFileAsync(hLib, pszFilePath,
                    TCS_DTYPE_UNKNOWN, TCS_SA_SCANONLY, 1,
                    NULL, OnTaskInProgressN);
    //Above call returns immediately.
    //Wait or do some other operations until scan is complete.
    TCSLibraryClose(hLib);

    return 0;
}
```

Implementation Guide

Performance and Resource considerations

For performance and memory considerations on mobile devices, it is recommended that creating library handle only when it is needed. To reduce the creation of the library handle, we can create the handle at the application initialization and release the handle when application quit. To use the minimal resource, we can share library handle between different threads, but caller need to provide thread safe by themselves. Here is an example about how to use CS library handle.

```
void ApplicationInit() {
            ... ...
            CsInit(context);
            ... ...
}

void ApplicationExit() {
            ... ...
            CsExit(context);
            ... ...
}

void CsInit(AppContext *context) {
            // set XM_HOME=${SDB file location}
            setenv("XM_HOME", "/usr/mcafee/xm_home");

            context->hLib = TCSLibraryOpen();
            if (context->hLib == INVALID_TCSLIB_HANDLE) {
                        // report error
            }
}

void CsExit(AppContext *context) {
            if (context->hLib != INVALID_TCSLIB_HANDLE) {
                        TCSLibraryClose(context->hLib);
            }
}

void ReportResult(TCSScanResult *pScanResult) {
            TCSDetected *pCur = NULL;

            if (int i = 0; pScanResult != NULL && i < pScanResult->iNumDetected; i++) {
                        ... ...
                        if (pCur == NULL) {
                                    pCur = pScanResult->pDList;
                        } else {
                                    pCur = pLast->pNext;
                        }
                        ReportInfection(pCur->pszName, pCur->pszVariant, pCur->pszFileName);
            }
}

int Scan(AppContext *context, const char *path) {
            int iRet = 0;
            TCSScanResult SR = {0};

            iRet= TCSScanFile(hLib,
                        path, // absolute full path to file
                        TCS_DTYPE_UNKNOWN, // scan engine to determine file type
                        1, // decompression required
                        TCS_SA_REPAIR, // repair
                        NULL, // private
                        NULL, // callback
                        &SR); // return result
            if (iRet == 0) {
                        ReportResult(&SR);
                        SR.pfFreeResult(&SR);
            }
            return iRet;
}

int ThreadSafeScan(AppContext *context, const char *path) {
            int iRet = 0;

            pthread_mutex_lock(&context->lock);
            iRet = Scan(context, path);
            pthread_mutex_unlock(&context->lock);

            return iRet;
`
```

Environment Variable Driven

Security vendors may allow caller to configure their environment by UNIX environment variable. For example, security vendor may need to configure the signature data base file path so that the content screening library can locate the signature data base when it gets created. Linux call setenv() can help caller to set this parameter to scan engine.

```
void CsInit(AppContext *context) {
            // set XM_HOME=${SDB file location}
            setenv("XM_HOME", "/usr/mcafee/xm_home");

            context->hLib = TCSLibraryOpen();
            if (context->hLib == INVALID_TCSLIB_HANDLE) {
                        // report error
            }
}

void CsExit(AppContext *context) {
            if (context->hLib != INVALID_TCSLIB_HANDLE) {
                        TCSLibraryClose(context->hLib);
            }
}
```

Thread Safe Coding

To make sure your code is thread safe, we need to make a good use of Linux pthread library, it provides a lot of thread safe functions for us to make our program better in muli-tasking application framework.

```
pthread_mutex_lock
pthread_mutex_trylock
pthread_mutex_unlock
pthread_cond_wait
pthread_cond_signal
pthread_cond_broadcast
```

```
void PutHandle(AppContext *context, hLib) {
    pthread_mutex_lock(&context->lock);
    PutHandleToPool(context, hLib);
    pthread_cond_signal(&context->cond);
    pthread_mutex_unlock(&context->lock);
    return hLib;
}

TCSLIB_HANDLE GetHandle(AppContext *context) {
    pthread_mutex_lock(&context->lock);
    pthread_cond_wait(&context->cond, &context->lock);
    hLib = GetHandleFromPool(context);
    pthread_mutex_unlock(&count_mutex);

    return hLib;
}
```

Error Code Demo

```
.
TCSLIB_HANDLE hLib;
TCSErrorCode ErrCode;
TCSScanResult ScanResult;
.
.
hLib = TCSLibraryOpen();
if (hLib == INVALID_TCSLIB_HANDLE)
{
   return( -1 );
}
.
.
if (TCSScanData(hLib, &ScanParam, &ScanResult) == 0)
{
.
.
    if (ScanResult.iNumDetected > 0)
    {
       // handle detections
    }
   ScanResult.pfFreeResult(&ScanResult);
}
.
.
TCSLibraryClose( hLib );
.
.
```

Possible Use Cases

As for email client, the scan can happen after the email get received, and before it gets put into user's inbox folder. Caller can pass either the email body or attachment or both of them separately to content screening scan data API to check if they are infected or not.

As for browser, the browser actually can pass web page to scan for malware before the web page gets rendered.

As for installer, it can request scan before the application installed on the device. For example, if we download the applications from market to "/download/apps/", installer can scan the file downloaded in this folder before copying it to user visible place "/user/apps".

As for application launcher the scan usually happens at the launch of the application. If the device was used without installing the scan engine, there could be some malware existing on the device even after user update their ROM to content screening security engine enabled. In this case, launcher can capture the malware which installer does not cover.

As for messenger, we usually focus on scanning the attachment of MMS, since it could carry malware in it. If the attachment is temporarily saved in some folder we can use file scan to scan it before showing it to user or we can scan the data as a whole before we save it to temporary file.

Caller	Data to be Scanned	Data Type	After receiving	Before storing	Before rendering	Before invoking	Before connect
Email Client	URL, HTML, Phone Number, Email Body	Text, HTML, Phone Number, Email	optional	optional	recommended		
Browser	HTML, JavaScript, embedded text (USSD)	Text, HTML, JavaScript	optional	optional	recommended		optional
Installer	HTML5	HTML	optional	optional		recommended	
Application Launcher	HTML5	HTML				recommended	
Messenger	SMS/MMS	Text, HTML, Phone Number, URL	optional	optional		recommended	