OpenClovis

OpenClovis Software Development Kit (SDK) Service Description and API Reference for Debug Service

For OpenClovis SDK Release2.3 V0.3

Document Revision Date: December 19, 2006

Contents

Chapter 1

Functional Overview

The OpenClovis Debug Infrastructure provides diagnostics access to all system components (including OpenClovis ASP service components, Componentized customer applications), irrespective of the location (node) where the component runs.

The CLI infrastructure routes the commands to the respective component which processes the command and provides response to the user. OpenClovis ASP service components offer CLI command handlers to client APIs.

The Debug infrastructure provides the following features:

- Using CLI commands, you can view or manipulate the data managed by OpenClovis ASP components. All these features are designed to assist the application developers and field engineers in testing and debugging applications.
- This CLI can be used for componentized customer applications. The CLI framework ensures that the application is accessible from the central CLI server. The actual CLI commands and the functionality are left for the application developers.
- The Debug CLI provides a centralized access to all OpenClovis ASP components and componentized applications. Using custom CLI commands a wide variety of services and features can be exposed to the CLI framework, which allows sophisticated, multi-component tests orchestrated through the debug CLI.

OpenClovis ASP comes with plug-ins for Ethereal, a popular packet sniffer tool, to capture, decode, and display OpenClovis ASP communication packets. These plug-ins are available for IOC, RMD, EM, NS, and GMS.

Initially in the debug CLI after establishing a connection, you are in OpenClovis ASP context. You can then list all the blades that are up and running and set context to a particular blade. Once you are in the blade's context, you can list all the components available on that blade and set context to a particular component. You can then start using the commands for that component. Once you are done working with a particular component, you can unset the current context and switch to other blades or components. When a command is entered at the terminal, it is sent to the gateway as is. The gateway parses the command and matches the command with the existing commands for the component for which the context is set. It then makes an RMD call to the debug object in the EO for that component. The debug object then matches the command to a function as per the registering done earlier and calls the appropriate function. The function is expected to return a string that is routed back to the terminal and displayed to the user.

Chapter 2

Service APIs

2.1 Type Definitions

2.1.1 ClDebugFuncEntryT

typedef struct {
ClDebugCallbackT fpCallback;
ClCharT funcHelp [CL_DEBUG_FUNC_HELP_LEN];
ClCharT funcName [CL_DEBUG_FUNC_NAME_LEN];
} ClDebugFuncEntryT;

The structure *ClDebugFuncEntryT* contains the entry for the debug CLI information that the component has to provide for every command.

- *funcHelp* - one line help for the command.
- *funcName* - name of the command used while invoking the command.stores the original file name.
- *fpCallback* - the function to be invoked upon a command entered on the CLI.

2.1.2 clEoExecutionObj

typedef struct {

textitClEoAppCreateCallbackT clEoCreateCallout;
ClEoAppDeleteCallbackT clEoDeleteCallout;
ClEoAppHealthCheckCallbackT clEoHealthCheckCallout;
ClEoAppStateChgCallbackT clEoStateChgCallout;
ClIocCommPortHandleT commObj;
ClEoIdT eoID;
ClUint32T eoInitDone;
ClOsalMutexIdT eoMutex;
ClIocPortT eoPort;
ClUint32T eoSetDoneCnt;
ClCntHandleT eoTaskIdInfo;
ClUint32T maxNoClients;

```
        ClCharT name [CL_EO_MAX_NAME_LEN];
        ClUint32T noOfThreads;
        ClEoClientObjT *pClient;
        ClCntHandleT pEOPrivDataHdl;
        ClOsalThreadPriorityT pri;
        ClUint32T refCnt;
        ClRmdObjHandleT rmdObj;
        ClEoStateT state;
        ClUint32T threadRunning;
} clEoExecutionObj;
```

The structure *clEoExecutionObj* contains the properties of an EO execution object. The execution object abstracts the properties of a running OS thread or process.

- *appType* - indicates whether application needs main thread or not.
- *clEoCreateCallout* - This application function is called from Main during the initialization process.
- *clEoDeleteCallout* - This application function is called when the EO gets terminated.
- *clEoHealthCheckCallout* - This is the application function that is called when EO health check is performed by CPM.
- *clEoStateChgCallout* - This is application function is called when the EO is moved into suspended state.
- *commObj* - This indicates the EO communication object.
- *eoID* - The eoID must be unique on a blade.
- *eoInitDone* - This indicates whether EOInit() has been called or not.
- *eoMutex* - This mutex is used to protect the Execution Object.
- *eoPort* - This indicates the requested IOC Communication Port.
- *eoSetDoneCnt* - This is used to set State related flag and counter.
- *eoTaskIdInfo* - This is the TaskID information of receive loop. It is used to delete the EO.
- *maxNoClients* - This is the maximum number of EO clients.
- *name[CL_EO_MAX_NAME_LEN]* - Execution object name.
- *noOfThreads* - This indicates the number of RMD threads spawned.
- *pClient* - This is the pointer to EO client APIs.
- *pEOPrivDataHdl* - This is the handle of the container of EO specific data.
- *pri* - This indicates the priority of the EO threads where RMD is executed.
- *rmdObj* - This is the RMD object associated with the EO. state This indicates the EO State.
- *threadRunning* - This is the receive loop thread State.

2.1.3 ClDebugPrintHandleT

typedef ClHandleT ClDebugPrintHandleT;

The type of the handle to be used for clDebugPrint APIs.

2.2 Library Life Cycle APIs

2.2.1 clDebugLibInitialize

clDebugLibInitialize

Synopsis:
Initializes the Debug CLI library.

Header File:
clDebugApi.h

Syntax:

```
ClRcT clDebugLibInitialize(void);
```

Parameters:
None.

Return values:
CL_OK: The API executed successfully.

Description:
This function is used as the library initialization routine for debug CLI. It must be called before executing any other debug functions.

Library File:
libClDebugClient.a

Related Function(s):
clDebugLibFinalize

2.2.2 clDebugLibFinalize

clDebugLibFinalize

Synopsis:
Finalizes the Debug CLI library.

Header File:
clDebugApi.h

Syntax:

```
    ClRcT clDebugLibFinalize(void);
```

Parameters:
None.

Return values:
CL_OK: The API executed successfully. And return values passed by EO APIs on an error.

Description:
This function is used as the library finalization routine for debug CLI. It is the last function called in order to do the necessary cleanup.

Library File:
libClDebugClient.a

Related Function(s):
clDebugLibInitialize

2.3 Functional APIs

2.3.1 clDebugCli

clDebugCli

Synopsis:
Invokes the library based local debug CLI.

Header File:
clDebugApi.h

Syntax:

```
ClRcT clDebugCli(
                    CL_IN ClCharT *nprompt);
```

Parameters:
nprompt: (in) String which is incorporated into the debug CLI prompt.

Return values:
CL_OK: The API executed successfully.

Description:
This function is used to invoke the library based local debug CLI. It will continue to execute till exited from the console. This function must be called from a separate thread.

Library File:
libClDebugClient.a

Related Function(s):
clDebugLibInitialize , clDebugLibFinalize

2.3.2 clDebugRegister

clDebugRegister

Synopsis:
Registers the component name.

Header File:
clDebugApi.h

Syntax:

```
            ClRcT clDebugRegister(
                                   CL_IN struct clEoExecutionObj* pEoObj,
                                   CL_IN ClCharT* compName,
                                   CL_IN ClCharT* compPrompt,
                                   CL_IN ClDebugFuncEntryT* funcArray,
                                   CL_IN ClUint32T funcArrayLen);
```

Parameters:

pEoObj: (in) Execution Object to register.

compName: (in) Name of the component to register.

compPrompt: (in) Prompt of the component to display in the CLI.

funcArray: (in) List of commands and their respective functions and help.

funcArrayLen: (in) Length of the function array.

Return values:

CL_OK: The API executed successfully.

CL_DEBUG_RC(CL_ERR_INVALID_PARAMETER): On passing an invalid parameter.

CL_DEBUG_RC(CL_ERR_NO_MEMORY): On failure to allocate memory.

Description:
This function is used to register the component name, prompt, and the list of all the CLI functions for the component with the EO. It is invoked before the debug server CLI can be used. The server debug CLI uses information registered by this function.

Library File:
libClDebugClient.a

Related Function(s):
clDebugDeregister

2.3.3 clDebugDeregister

clDebugDeregister

Synopsis:
De-registers the debug CLI information from the EO.

Header File:
clDebugApi.h

Syntax:

```
ClRcT clDebugDeregister(
                        struct clEoExecutionObj* pEoObj);
```

Parameters:
pEoObj: (in) Execution Object to deregister.

Return values:
CL_OK: The API executed successfully. And other errors returned by EO APIs.

Description:
This function is used to de-register the debug CLI information from a given EO.

Library File:
libClDebugClient.a

Related Function(s):
clDebugRegister

2.3.4 clDebugPrintInitialize

clDebugPrintInitialize

Synopsis:
Retrieve a handle for printing.

Header File:
clDebugApi.h

Syntax:

```
ClRcT clDebugPrintInitialize(
                    CL_OUT ClDebugPrintHandleT* msg);
```

Parameters:
msg: (out) Handle for printing.

Return values:
CL_OK: The API executed successfully.

CL_DEBUG_RC(CL_ERR_NO_MEMORY): On failure to allocate memory. And other errors returned by EO APIs.

Description:
This function is used to retrieve a handle for printing.

Library File:
libClDebugClient.a

Related Function(s):
clDebugPrintFinalize, clDebugPrint

2.3.5 clDebugPrint

clDebugPrint

Synopsis:
Prints a string into the handle.

Header File:
clDebugApi.h

Syntax:

```
ClRcT clDebugPrint(
                  CL_INOUT ClDebugPrintHandleT msg,
                  CL_IN const char* fmtStr,
                  ...);
```

Parameters:

msg: (in/out) Handle for printing.

fmtStr: (in) Format string for the variadic arguments.

vargs: Variadic arguments.

Return values:

CL_OK: The API executed successfully.

Description:
This function is used to print a string with a maximum of 512 bytes into the handle at any given point in time. This function can be invoked 'n' number of times after invoking *clDebugPrintInitialize* function, before calling either *clDebugPrintFinalize* or *clDebugPrintDestroy* functions.

Library File:
libClDebugClient.a

Related Function(s):
clDebugPrintInitialize, clDebugPrintFinalize

2.3.6 clDebugPrintFinalize

clDebugPrintFinalize

Synopsis:
Cleans up the print handle.

Header File:
clDebugApi.h

Syntax:

```
ClRcT clDebugPrintFinalize(
                    CL_IN  ClDebugPrintHandleT* msg,
                    CL_OUT char** buf);
```

Parameters:

msg: (in) Handle for printing.

buf: (out) Buffer containing the print message.

Return values:

CL_OK: The API executed successfully.

CL_DEBUG_RC(CL_ERR_NULL_POINTER): On passing a NULL pointer.

CL_DEBUG_RC(CL_ERR_NO_MEMORY): On failure to allocate memory.

Description:
This function is used to clean up the print handle. It returns a memory allocated buffer containing the print messages logged into the handle.

Library File:
libClDebugClient.a

Related Function(s):
clDebugPrintInitialize

2.3.7 clDebugPrintDestroy

clDebugPrintDestroy

Synopsis:
Frees the print handle.

Header File:
clDebugApi.h

Syntax:

```
ClRcT clDebugPrintDestroy(
                 CL_INOUT ClDebugPrintHandleT* msg);
```

Parameters:

msg: (in/out) Handle for printing.

buf: Buffer containing the print message.

Return values:

CL_OK: The API executed successfully. It returns all error codes returned by *clBufferMessageDelete()*.

Description:
This function is used to free the handle without returning any message buffer.

Library File:
libClDebugClient.a

Related Function(s):
clDebugPrintInitialize

2.3.8 clDebugVersionCheck

Synopsis:
Check the given version is supported or not.

Header File:
clDebugApi.h

Syntax:

```
ClRcT clDebugVersionCheck(
                        CL_INOUT ClVersionT* pVersion);
```

Parameters:

pVersion: (in/out) the version.

Return values:

CL_OK: The API executed successfully. It returns all error codes returned by clVersion-Verify().

Description:
This function is used to check the given version is compatible or not. If not, will return the compatible version with error.

Library File:
libClDebugClient.a

Related Function(s):
clDebugLibInitialize , clDebugLibFinalize

Index