Sustainable Software Concepts

SYNTHESIS AG

www.synthesis.ch

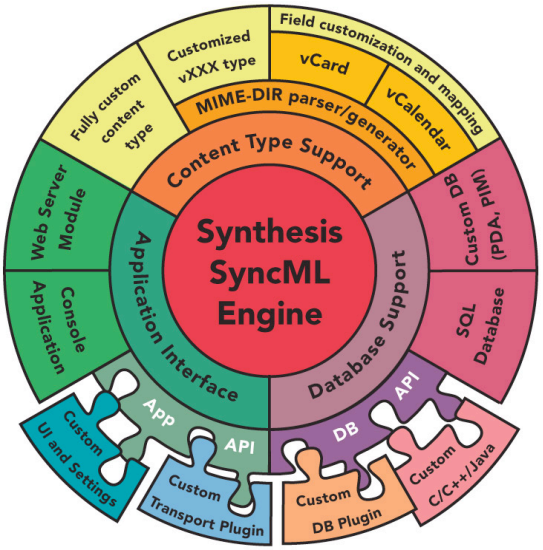
# Reference Manual for

# SDK and Plugin Interface V1.9.2

# of the Synthesis Sync Engine V3.4

24-Jun-2011

SyncML



### This manual was written for Synthesis SyncML Engine V3.4



Synthesis SyncML Engine uses parts of the following software:

### expat - XML parser - http://sourceforge.net/projects/expat



### SyncML toolkit - http://sourceforge.net/projects/syncml-ctoolkit/



### zlib compression library - http://www.zlib.net/



### SQLite 3 database engine - http://www.sqlite.org/

### PCRE Library - http://www.pcre.org/license.txt



The project files to create the SySync SDK plug-ins are using the following software:

### C/C++ CodeWarrior compiler environment - http://www.metrowerks.com



### Visual Studio - http://www.microsoft.com



### XCode - http://developer.apple.com/tools/xcode



### Android - http://www.google.com/mobile/android



### Disclaimer

# Contents

# 1. Introduction

Thank you for choosing **Synthesis Sync Server/Client** as your SyncML solution. It provides you with a very efficient compliant SyncML engine with many advanced features and especially a high configurability.

Synthesis Sync Server/Client exists in different versions for different database interfaces.

This manual covers Synthesis SyncML products supporting custom plugins for interfacing with the database (**DB Api**), like the Synthesis SyncML Servers in the PRO version as well as Synthesis SyncML library products which come as a loadable library (.dll, .so, .dylib) and have a API to access the SyncML functionality from a client application (**UI Api**) .
Custom plugins and applications can be written in **C/C++, C#** and **Java** as well as in any programming language capable of the C-style calling conventions (e.g. Borland Delphi).

This manual contains the reference for the Software Development Kit (**SDK**) required to create both custom database plugins and applications.

This manual does not cover configuration of the SyncML engine itself. Please refer to the SySync_config_reference.pdf manual which is part of the SDK package and most server product packages.

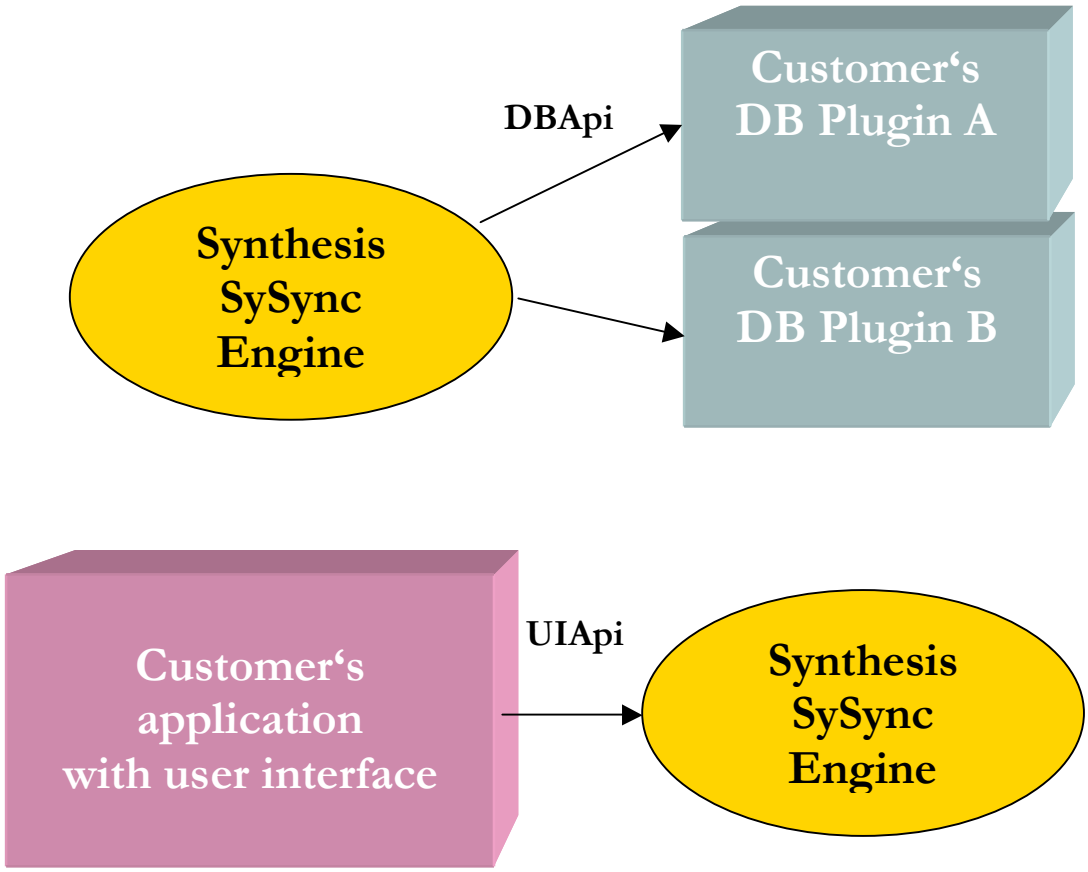
# 2. Overview

Synthesis AG makes their SyncML engine functionality available for customized database plug-in adapters as a Software Development Kit ( **SDK** ). Synthesis Plugin technology allows the customer to develop data base adapters or user interfaces without the need of understanding the details of the SyncML standard. It's an ideal division of work between Synthesis and the customer's project: Synthesis delivers a scalable, high performance SyncML OMA DS 1.2 engine, which is interoperability-tested against a huge variety of SyncML devices on the market. The customer only needs the specific knowledge to access his own data base framework or his own user interface which can be written in several programming languages.
A small interface with only 48 + 23 well documented and easy-to-use functions is the bridge of interaction. All SyncML protocol details are hidden.

There are mainly two sections of the SDK:

- The **data base interface** ( **DBApi** ) for writing data base plugins (see chapter 4).
- The **user interface** ( **UIApi** ) for writting user interfaces (see chapter 5).

Both sections can be used completely independently, though some interface files are shared.



- Programming interface for **C/C++**.
- A plug-in for access to **Java** thru JNI (Java Native Interface) is also available.
- The UI and DB interfaces for **C#** are available (since version V1.4.0).
- The UI and DB interfaces for **Delphi** are available ( since version V1.4.0).
- Other interfaces will be implemented on request.
- „Ready to use“ example code for a demo database module in C, a „textdb“ interface in C++, the OceanBlue/SnowWhite example adapter in C++ and a demo module in Java, C# and Delphi are part of the package to demonstrate the DBApi (see chapter 4).
- „Ready to use“ complete SyncML client examples for Mozilla sunbird calendar for Windows/Delphi, MacOSX/Cocoa/XCode and Linux are included to demonstrate the UI Api (see chapter 5). Several small sample applications to demonstrate specific aspects of the UIApi are also available.
- **Windows**, **Linux, iPhoneOS**, **MacOSX** and **Android** target platforms are supported at this time. For development, Metrowerks' CodeWarrior project files are available, as well as Visual Studio 2005 vcproj file for Windows, XCode project file for Mac OSX and iPhoneOS, a makefile for Linux and the Eclipse environment for Android.
- Versions for Windows Mobile and other platforms are planned for the future.
- The code can be compiled by the customer as an application for the UI application and as a Dynamic Link Library ( DLL ) for the DB Api plugins.
- Multiple plug-ins can be used in parallel at the same time.
- The SDK allows multi-threading to support multiple simultaneous sessions of the SyncML server.
- Easy configuration via the main XML configuration file
- There is no specific version of the Synthesis SyncML Server/Client with the plug-in technology, all future servers will contain it. Only the license decides, whether the functionality can be used or not.
- The Synthesis demo server and client contain the current version of „SDK_textdb“ as a built-in plug-in.

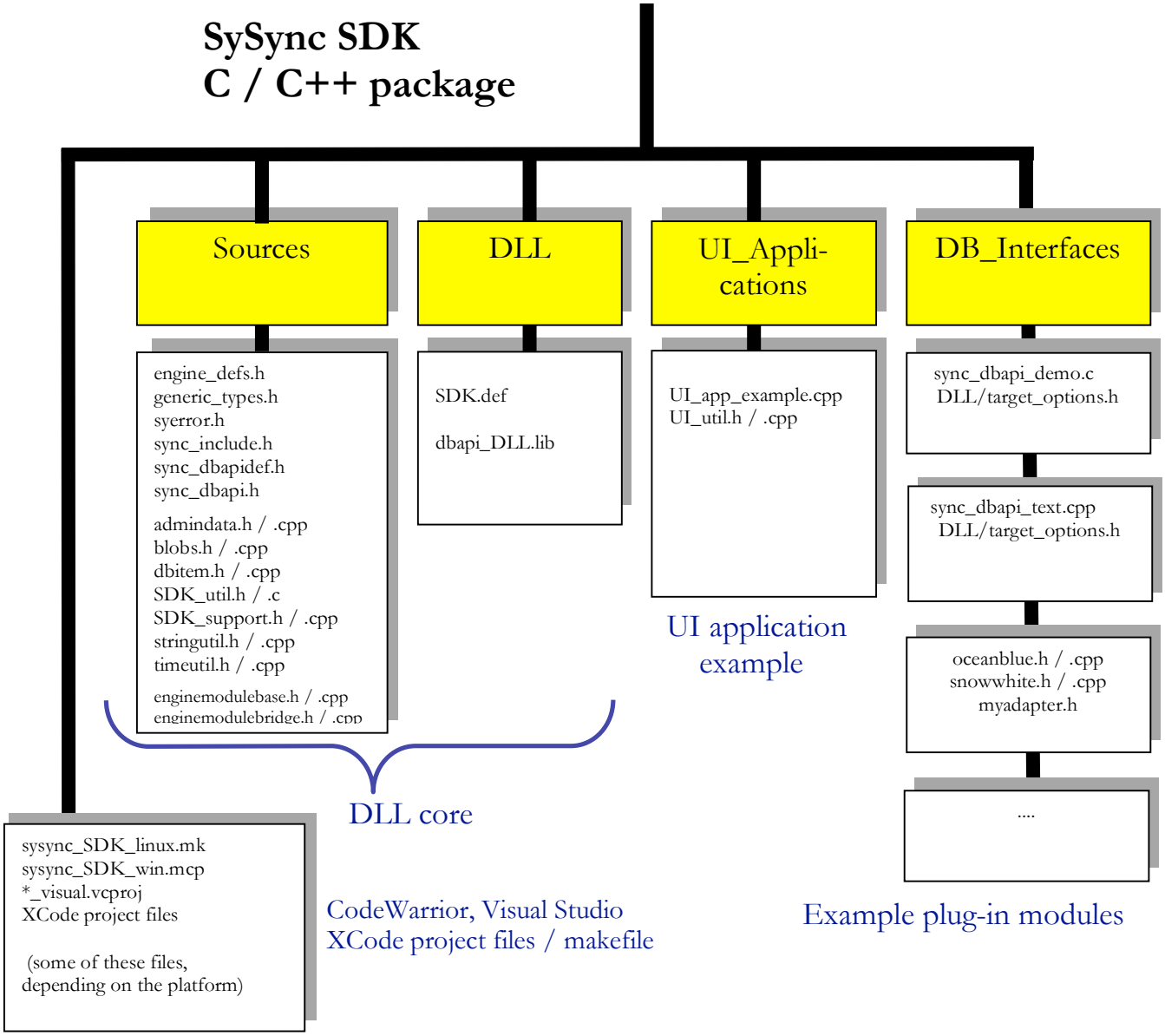
With the SySync **DBApi SDK**, the customer is able to create plug-ins, which will be called directly by the Synthesis SyncML engine. The SyncML engine acts as a master: It makes subroutine calls into the plugin DLL. Each routine must return an error status, which will be handled by the engine. The main three blocks are the **Module**, the **Session** and the **Datastore** handling. These three blocks are normally kept within one DLL, but they can be separated into different DLLs as well. The description of each routine with several programming hints can be found in the interface definition file „sync_dbapi.h“. Note that the DBApi plugin interface is not available in all variants of Synthesis SyncML client libraries (such as the free DEMO variants).

The access to all blocks is **context based**, so at the beginning a routine „Create_XXX“ will be called, which has to return a unique identifier which will be used for each subsequent call of this context. The „Delete_XXX“ will remove this context again later. Variables which are local within such a context must be stored within this environment during its lifetime. This can be done either by using the context identifier as pointer to a local structure or using it as an index.

With the SySync **UIApi SDK**, the customer can write his own user interface and communication code (in the current version for SyncML available for client applications) and is calling the SyncML engine for **initialisation**, **syncing**, **message reading/writing** and **parameter setup**. In this configuration, the UI application is usually a program which acts as master and is using the SyncML engine as shared (or linked) library.

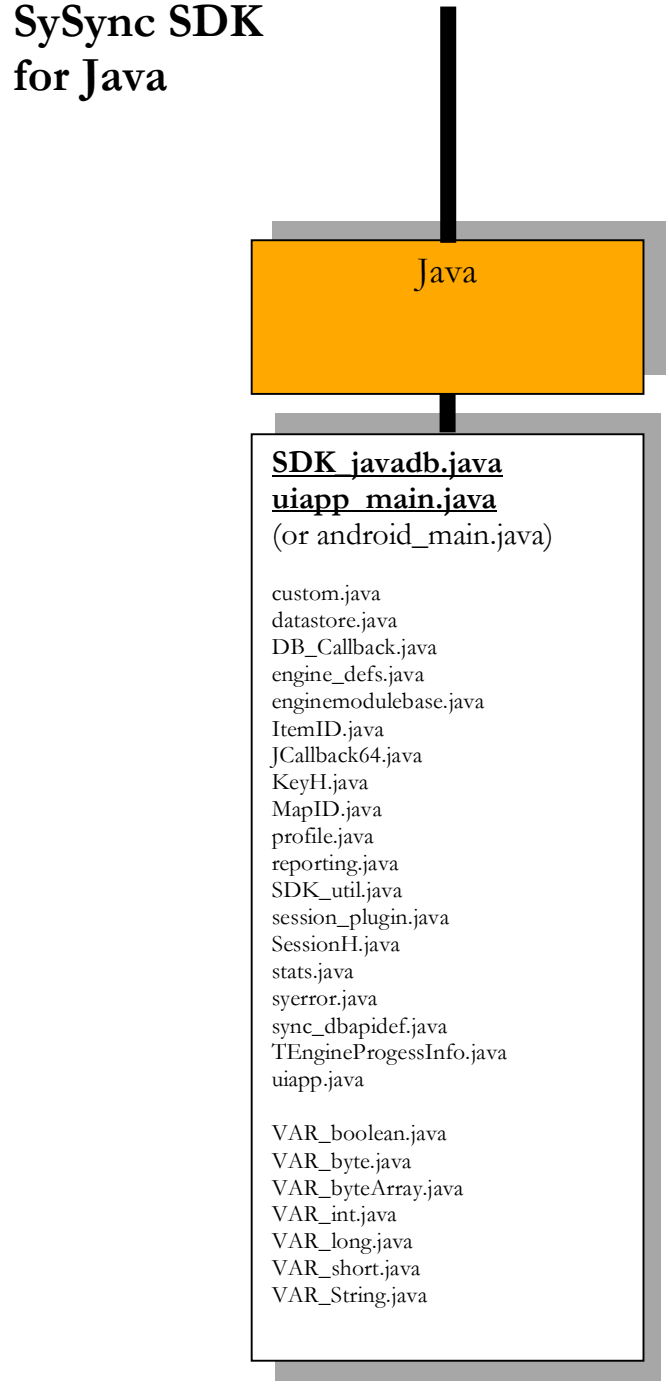
# 3. Distribution Files

The distribution media (normally a .ZIP archive) contains the following files
(NOTE: depending on the version you have, not all of the listed files will be included):




There are other subdirectories with specific projects examples for C#, Delphi, XCode (sunbird client example), ....

The JNI / Java files can be found at:




All SDK platforms with Java interface are now based on a „**com.sysync**“ package of the Java environment.

The Java SDK for MacOSX, Windows and Linux for earlier versions has been provided within the „sysync“ package and outside any package. From V1.6.1 onwards, **all platforms** are based now on the package „**com.sysync**“ using **64 bit (long) signatures** for both UI and DB adapters, the former entry points to the engine however are still available.

The Java SDK contains an **example** of a **DB adapter** example without functionality (comparable to the SDK_demo adapter, written in C language): **SDK_javadb.java**.

There is also a Java example which can be used as starting point for a sync **UI application**: **uiapp_main.java** (or **android_main.java**).

The other java modules of the package contain:

- engine_defs, syerror, sync_dbapidef:
  const definitions of the engine, which are equivalent to the corresponding C/C++ files.
- VAR_*: function/method parameters of scalar types which can be changed inside (equivalent to VAR parameter in Pascal or & paramters in C++)
- JCallback64: The calling interface to the engine
- DB_Callback, KeyH, ItemID, MapID, SessionH, TEngineProgressInfo:
  The structured types of the Callback/Call-In methods
- enginemodulebase/SDK_util:
  Some utility functions, the mainly help to use GetValue/SetValue directly for several integer, string and buffer types
- custom, datastore, profile, reporting, session_plugin, stats:
  More utility structures and methods
- uiapp: The example code for the UI application, called from uiapp_main (or android_main)

There are two direct entry points each to the engine (as client or as server)

```
// direct calls to the Synthesis SyncML engine (as client)
  static native short     SySync_ConnectEngine( DB_Callback aCB, VAR_int aEngVersion,
                                                                     int aPrgVersion,
                                                                   short aDebugFlags,
                                                                  String jClassName );
  static native short     SySync_DisconnectEngine( long aCB );


  // direct calls to the Synthesis SyncML engine (as server)
  static native short SySync_srv_ConnectEngine( DB_Callback aCB, VAR_int aEngVersion,
                                                                     int aPrgVersion,
                                                                   short aDebugFlags,
                                                                  String jClassName );
  static native short SySync_srv_DisconnectEngine( long aCB );
```

Please note, that the server engine is not yet available for all SDK platforms.
The server entry points are **only** available for the **com.sysync** package.

For clients also the com.sysync package entry points should be used, the former entry points are however still available for downwards compatibility reasons.

# 4. SySync DBApi SDK description

The main three blocks of the SySync Software Development Kit (SDK) are the **Module**, the **Session** and the **Datastore** handling. These three blocks are normally kept within one DLL, but they can be separated into different DLLs as well. The description of each routine with several programming hints can be found in the interface definition file „sync_dbapi.h“ and in chapter 7 of this manual. Here is an overwiew over the routines of these three main blocks:

## 4.1 How to write a database plugin ?

After having chosen the programming language for the plugin ( C, C++, C#, Java, Delphi ), the best starting point is to take the dbapi example and add the specific functionality. Not all functions must be implemented at all or right from the beginning, replacement can be done step by step. The calling direction is always from SyncML engine to the plugin module and returning afterwards to the SyncML engine (usually with an error code). The DBApi plugin has **identical** structure for SyncML **servers** and **clients**, so the same plugin module can be used on both sides. For C++ programming, a good starting point is the OceanBlue / SnowWhite adapter, see chapter 4.7.

It is recommended to use the callback debug output system, which is already part of all example files. So the DBApi plugin will write the flow information directly into the log file.

A good starting point is the implementation and adaption of the **module** context which must return information of the plugin module to the engine. All basic information is already implemented at the example files.

In a second phase the **session** context is needed for assigning user and devices, here a minimum setup for login handling is requested.

The most important part is the **datastore** context handling where the user data will be read and written. The admin section needn't to be implemented for every database plugin, as it can be handled by a different module as well (the config file must contain the appropriate info for this).

A detailed descriptions of these context systems is described in the next chapters.

## 4.2 Module Handling

- Module_CreateContext
- Module_Version
- Module_Capabilities
- Module_PluginParams
- Module_DisposeObj *)
- Module_DeleteContext

*) Not implemented for JNI and C#, because Java and C# run their own garbage collection

This is the set of routines for the Plug-In access on the module level. When the SyncML engine connects to a Plug-In module, „Module_CreateContext“ will be called first.When disconnecting, „Module_DeleteContext“ will be called as the final call. The SyncML engine will create a module context for the sessions and one for each datastore admin and data section.
„Module_CreateContext“ can either create a new context or share a global module context among session and datastores. Module context „0“ is reserved.

„Module_Version“ and „Module_Capabilities“ inform the engine, what is currently supported within the plug-in module. With „Module_PluginParams“ the SyncML engine informs the plug-in module about <plugin_params> of the XML config file.
The plugin must be able to return „Module_Version“ of context „0“ without any preceding „Module_CreateContext“. The module version cannot be defined by the plugin programmer, as it contains compatibility information for the engine. The only thing the user can define is the build number 0..255.

„Module_Capabilities“ can return **NoField** identifiers ( example: „plugin_sessionauth:no“ ) which allows to remove some DLL functions completely, not even the entry points must be available then. This is also true for the Java environment where these methods needn't to be implemented, if switched off. For C# all functions must be available.

**Supported NoField sections**:

| | | |
|---|---|---|
| - Plugin_Session | „plugin_se:no“ | (the whole session) |
| - Plugin_SE_Adapt | „plugin_sessionadapt:no“ | (session adaptitem) |
| - Plugin_SE_Auth | „plugin_sessionauth:no“ | (session login) |
| - Plugin_DV_Admin | „plugin_deviceadmin:no“ | (session admin) |
| - Plugin_DV_DBTime | „plugin_dbtime:no“ | (session „GetDBTime“) |
| | | |
| - Plugin_Datastore | „plugin_ds:no“ | (the whole datastore) |
| - Plugin_DS_Admin | „plugin_datastoreadmin:no“ | (admin part) |
| - Plugin_DS_Data | „plugin_datastore:no“ | (data part) |
| - Plugin_DS_Data_Str | „plugin_datastore_str:no“ | (data part as str) |
| - Plugin_DS_Data_Key | „plugin_datastore_key:no“ | (data part as key) |
| - Plugin_DS_Blob | „plugin_datablob:no“ | (BLOB support) |
| - Plugin_DS_Adapt | „plugin_dataadapt:no“ | (data adaptitem) |

The **plugin_info program**, which is part of the SDK package, shows the feedback about these informations.

**NOTE:** The admin part requires also BLOB support for SyncML 1.2. That's because an incomplete item during suspend/resume will be stored as BLOB.

„Module_DisposeObj“ asks for deallocation of memory (which has been allocated within the module to get the capabilities string).

## 4.3 Session Handling

These routines handle the session context at a plug-in module. Main tasks of this blocks are device info & nonce handling and the user authentification (login). The return values of this block will be used later to access the datastores.

**NOTE:** The Session_PasswordMode mode must be in line with the config file's authentification settings.

**NOTE:** The **client** side requires only a **rudimentary session handling**. E.g. for Java clients the session_plugin.java module can be used: It's mainly switching the datastore debug callback to the session log file. But no user/login or nonce handling is usually needed on the client side.

Multiple sessions can run in parallel, using the concept of multi-threading at the SyncML engine. Therefore all operations MUST refer only to the <sContext> variable ( which must be created in the plug-in function „Session_CreateContext" and deleted with „Session_DeleteContext").
The SyncML engine will never call a context again after „Session_DeleteContext", it assumes that all allocated resources of the session are removed there.

Interference between sessions should be avoided or must be made thread-safe. Even the thread of a running session can change: The SyncML engine will give a notification before such a change by calling the routine „Session_ThreadMayChangeNow". As the name says, it may change (but it must not). If this information is not needed for the plugin module, it can be implemented empty.

- Session_CreateContext
- Session_AdaptItem 1)

- Session_CheckDevice 2)
- Session_GetNonce 2)
- Session_SaveNonce 2)
- Session_SaveDeviceInfo 2)

- Session_GetDBTime 3)

- Session_PasswordMode 4)
- Session_Login 4)
- Session_Logout 4)

- Session_ThreadMayChangeNow
- Session_DisposeObj 5)
- Session_DispItems 6)
- Session_DeleteContext

1) Needn't to be implemented with „plugin_sessionadapt:no" at Module_Capabilities

2) These routines will be called only, if <api_deviceadmin> is set to yes at the config
Needn't to be implemented with „plugin_deviceadmin:no" at Module_Capabilities

3) Needn't to be implemented with „plugin_dbtime:no" at Module_Capabilities

4) These routines will be called only, if <api_sessionauth> is set to yes at the config
Needn't to be implemented with „plugin_sessionauth:no" at Module_Capabilities

5) Not implemented for JNI and C#, because Java and C# run their own garbage collection
6) Will never be called by the SyncML engine; for debug purposes only.

## 4.4 Datastore Handling

A datastore will always be accessed within a session. Multiple datastore accesses within a session will not run sequentially, they can even run in parallel to other sessions. The datastore handling has always the same flow: open – [admin read ] - read – update – [admin write ] - close. Therefore the datastore handling is divided into several sub sections. Detailed description can be found at „sync_dbapi.h“.

NOTE: For each datastore **two separate contexts** will be opened for the **admin** and the **data** part. This is because they can be handled by two separate plugin modules or one of them as ODBC, the other one as plugin. So they will be handled separately even if they are using the same plugin module.To distinguish which one is which, the engine can be configured (by returning „**ADMIN_Info:yes**“ with „Module_Capabilities“) to add the word „ ADMIN“ to <aContext-Name> of „CreateContext“ when called as admin context.

### *4.4.1 The “open” section*

The „open“ section will „Create_Context“ and provides context and filter options to the SyncML engine.

- CreateContext
- ContextSupport
- FilterSupport"

**NOTE:** „ContextSupport“ and „FilterSupport“ calls will appear usually at the beginning of the data store handling, but under certain conditions they can be called at any time during the datastore handling. Multiple calls are possible.

Example:

FilterContext call 1:
```
daterangestart:20070219T191809Z
daterangeend:20070619T191809Z
```

FilterContext call 2:
```
staticfilter:
dynamicfilter:
invisiblefilter:F.SYNCLVL:=0|F.SYNCLVL*=E
```

FilterContext call 1 will pass the /dr(-before/after) conditions (as ISO8601 time) to the plugin. The field names are predefined, for details see also the filter section at the Synthesis Config Reference manual. The plugin should return 2 (for 2 supported fields), if both values are supported and fully considered. It should return 0, if they are not or partly considered. E.g. a plugin might be able to filter only on date resolution, so it can make this raw prefiltering. By returning 0, the engine will make still the fine filtering.

FilterContext call 2 will switch off staticfilter and dynamicfilter and will try to install the invisiblefilter. If the plugin supports invisible filtering, it should return the value 3 (for 3 supported fields). datarangestart/datarangeend are **not affected** with the 2nd call, so they are still active.

### 4.4.2 The "admin read" section

The „admin read" section allows to handle the map tables. Detailed description can be found in „sync_dbapi.h". All routines of this section can be implemented empty with return code DB_Forbidden = 403, if the admin tables will be handled by the SyncML engine itself.
There is even a way to remove these routines completely.

**NOTE:** For Windows, the according entry points must be removed from the „.def" file.

- LoadAdminData
- ReadNextMapItem

These routines will be called only, if <plugin_datastoreadmin> is set to yes in the config.
It needn't to be implemented with „plugin_datastoreadmin:no" at Module_Capabilities

**NOTE:** Some of the Synthesis SyncML (client) engines have the admin part **built-in**, so it cannot be redirected to a plugin module for these cases.

### 4.4.3 The "read" section

The „read" section starts with „StartDataRead" and ends with „EndDataRead".

- **StartDataRead**
- ReadNextItem *)
- ReadNextItemAsKey *)
- ReadItem *)
- ReadItemAsKey *)
- ReadBlob **)
- **EndDataRead**

*) Needn't to be implemented with „plugin_datastore:no" at Module_Capabilities
**) Needn't to be implemented with „plugin_datablob:no" at Module_Capabilities

„ReadNextItem"/ „ReadItem": <aItemData> returns the data, formatted as multiline, where <aa> / <cc> are the **identifiers** and <bb> / <dd> the **data fields**:
The field **separator** generated by the engine is <CRLF> = \r\n = 0x0d 0x0a..
( The engine is able to handle <CR> only, as well as <LF> only as separator )

aa:bb<CRLF>cc:dd[<CRLF>]

The identifiers are either assigned to the fieldmap names, or just numbered by an index, if automap **indexasname** is true:

**Example:**
<aItemData>:
0:Joe<CRLF>1:Smith<CRLF>2:New York<CRLF>

with XML config file entry:

```
<fieldmap fieldlist="calendar">
    <automap indexasname="true"/>
</fieldmap>
```

The „SDK_textdb“ sample is expecting indexasname=“true“, because the fieldmap names will not be stored, so the ordering of the config file's fieldmap determines the index assignment.
**NOTE:** Adding fields in-between or changing the ordering of fields will make the system in-compatible to already existing TDB_*.txt files, when indexasname is true.

The data fields can be multiline, so carriage returns <CR> must be escaped using „\r“, linefeeds <LF> must be escaped using „\n“. To allow this, also backslashes themselves must be escaped (using „\\“). Double quotes and ctrl characters must be escaped as well. For details see the string conversion routines at „stringutil.cpp“, which is part of the SDK package.

There are two extensions to this syntax:

- **BLOBs:** For binary large object blocks the field contains only a **reference** to the BLOB identifier which will be read and written with ReadBlob/WriteBlob.
  Syntax: aa;BLOBID=xyz  where <xyz> is the name of the BLOB.

- **Arrays:** For array fields a syntax with index will be used
  Syntax: aa[index]:bb

„ReadNextItemAsKey“ and „ReadItemAsKey“ are equivalent to „ReadNextItem“/ „ReadItem“, but they are using an appPointer <aItemKey> instead of transferring the <aItemData>. They will be used **instead** by the SyncML engine, if „ITEM_AS_KEY:true“ is returned with Module_Capabilities and at least SDK 1.4.0 is used. These keys are completely opaque for the plugin module. Their attached context must be read or written with the GetValue/SetValue callback functions.

### 4.4.4 The “update” section

The „update“ section starts with „StartDataWrite“ and ends with „EndDataWrite“. Read commands ( ReadItem / ReadBlob ) can appear here as well.

- **StartDataWrite**
- InsertItem *)
- InsertItemAsKey *)
- FinalizeLocalID
- UpdateItem *)
- UpdateItemAsKey *)
- MoveItem
- DeleteItem
- DeleteSyncSet
- WriteBlob **)
- DeleteBlob **)
- **EndDataWrite**

*) Needn't to be implemented with „plugin_datastore:no“ at Module_Capabilities
**) Needn't to be implemented with „plugin_datablob:no“ at Module_Capabilities

„InsertItemAsKey“ and „UpdateItemAsKey“ are equivalent to „InsertItem“/ „UpdateItem“, but they are using an KeyH <aItemKey> instead of transferring the <aItemData>.
They will be used **instead** by the SyncML engine, if „ITEM_AS_KEY:true“ is returned with Module_Capabilities and at least SDK 1.4.0 is used. These keys are completely opaque for the

plugin module. Their attached context must be read or written with the GetValue/SetValue callback functions.

**NOTE:** „MoveItem“ is prepared for handling hierarchical datastores. In the current version the SyncML engine has not yet implemented this feature. Therefore this function will not yet be called. For current plugin implementations LOCERR_NOTIMP (20030) can be returned.

### 4.4.5 The “admin write” section

The „admin write“ section allows to handle the map tables. Detailed description can be found at „sync_dbapi.h“. All routines of this section can be implemented with return code DB_Forbidden = 403, if the admin tables will be handled by the SyncML engine itself.

- SaveAdminData
- InsertMapItem
- UpdateMapItem
- DeleteMapItem

These routines will be called only, if <plugin_datastoreadmin> is set to yes in the config.
Needn’t to be implemented with „plugin_datastoreadmin:no“ at Module_Capabilities

**NOTE:** Some of the Synthesis SyncML (client) engines have the admin part **built-in**, so it cannot be redirected to a plugin module for these cases.

### 4.4.6 The “general” section

Some general routines are part of this section:

- ThreadMayChangeNow
- WriteLogData
- AdaptItem *)
- DisposeObj **)
- DispItems ***)

*) Is not yet implemented in the SyncML engine, so it will never be called.
Needn’t to be implemented with „plugin_dataadapt:no“ at Module_Capabilities
**) Not implemented for JNI and C#, because Java and C# run their own garbage collection
***) Will never be called by the SyncML engine; for debug purposes only.

As the name says, the thread may change after „ThreadMayChangeNow“ (but it must not).
If this information is not needed for the plugin module, it can be implemented empty.

### 4.4.7 The “close” section

The „close“ section releases the data store. The plug-in must release here all allocated memory for this datastore. All objects returned to the SyncML engine will be released with ‚DisposeObj’ prior this call.

- DeleteContext

No access to this context will be done after „DeleteContext“.

## 4.5 Callback calls

The Synthesis SyncML engine supports a callback mechanism, which can be used at the plug-in modules for writing comments to the log files. Logging will be done on module, session and datastore level. The user should **NEVER** use „printf“ or „cout“ calls, as this kind of output is not supported by all versions of the Synthesis SyncML server and will not be logged in an appropriate way. The SDK_util file provides the *DEBUG_Call* and the *DEBUG_DB* call:

```
void DEBUG_Call( void* aCB, uInt16 debugFlags,
                 cAppCharP ident, cAppCharP routine,
                 cAppCharP text, ... );

void DEBUG_DB  ( void* aCB,
                 cAppCharP ident, cAppCharP routine,
                 cAppCharP text, ... );
```

*DEBUG_DB* is a *DEBUG_Call* with <debugFlags> = DBG_PLUGIN_DB
The <aCB> variable will be passed with the creation of each context (and must be stored within the context object for subsequent use). The SyncML engine will write the text to the context assigned log file. For more details see descriptions at „sync_dbapi.h“.

Sometimes, very extensive logging is requested, which should not be visible in normal log files. The SyncML engine supports a flag called <exotic>.
Calls of *DEBUG_Exotic_Call* or *DEBUG_Exotic_DB* will be shown only, if the global "exotic" debug flag is set:

```
void DEBUG_Exotic_Call( void* aCB, uInt16 debugFlags,
                        cAppCharP ident, cAppCharP routine,
                        cAppCharP text, ... );

void DEBUG_Exotic_DB  ( void* aCB,
                        cAppCharP ident, cAppCharP routine,
                        cAppCharP text, ... );
```

*DEBUG_Exotic_DB* is a *DEBUG_Exotic_Call* with <debugFlags> = DBG_PLUGIN_DB

The log file can be structured using logical blocks.
To add these structures, use DEBUG_Block / DEBUG_EndBlock as pairs. <aTag> identifies such a pair.

```
void DEBUG_Block   ( void* aCB, cAppCharP aTag, cAppCharP aDesc,
                                                cAppCharP aAttrText );
void DEBUG_EndBlock( void* aCB, cAppCharP aTag );
```

The end of a thread can be signalled with. This information helps to create more structured logs.

```
void DEBUG_EndThread( void* aCB );
```

**NOTE:** The underlying debug callback calls are using cb->callbackRef as first parameter, not <aCB> directly. If you are using these calls directly (e.g. on language platforms where SDK_util is not available), please be aware that debug callback calls and UI callin calls are treated differently concerning this first parameter.

## 4.6 The global context

There are two main reasons to have a global context: Either 1) for some reasons **no global variables** are allowed within the plugin module or 2) there is a need to **share** some variables between different plugin modules.

For both cases the SyncML engine provides a mechanism to get such a global context without the need of global variables. A structure „GlobContext“ (defined at „sysync_dbapidef.h“) will be provided at „Module_CreateContext“ thru **mCB->gContext**.

```
/*! Structure of GlobContext */
struct GlobContext {
  void*               ref;       /* reference field */
  struct GlobContext* next;      /* reference to the next GlobContext structure */
  uInt32              cnt;       /* link count */
  char                refName[ 80 ]; /* the reference's name, length restricted */
};
```

<refName> which is initially empty can be assigned any specific name of this context and <ref> should point to the desired global structure. The <cnt> must be incremented by 1.
The <next> field needn't to be handled, this will be done by the SyncML engine.

mCB->gContext actually points to a linked list of GlobContext, where <next> points to next element, as long as not NULL.

In subsequent calls of „Module_CreateContext“ it can be searched for the specific name at this linked list. If available, <ref> is the desired reference. Don't forget to increment <cnt> for each assigned reference.

Each module context with such a reference must decrement <cnt> at „Module_DeleteContext“ again. When <cnt> reaches 0, the reference structure should be deleted, then <ref> set to NULL and <refName> to „“. The SyncML engine will automatically remove such empty elements.

Each plugin module can use up to 3 such GlobContexts with a different <refName>.
The function „GlobContextFound“ can be used to search/assign such a GlobContext.

An example for a global structure which will be used by different plugin modules is a reference to a virtual machine which only exists once per system. The Java Bridge „JNI“ is built with such a reference to the JavaVM.

## 4.7 The OceanBlue / SnowWhite adapter

For C++ an example implementation with a base class ( „OceanBlue“ ) and a derived class („SnowWhite“ ) is part of the package. „OceanBlue“ contains all interface function as virtual methods which can be overriden by the „SnowWhite“ classes. The given example implements version and capablity feedback for the module level, login for the session level and the data handling methods readnext/read/insert/update/delete with some example code.

The intension is that „OceanBlue“ must not be changed, all adaptions will be done at the „SnowWhite“ module. Here is a **step-by-step tutorial** how to create your own database adapter.

1) Make a copy of „snowwhite.h“ and „snowwhite.cpp“ for creating your own database adapter.

2) Adapt the name „snowwhite“ at „myadapter.h“ to your own plugin's name. The snowwhite sources do not contain „SnowWhite“ directly, they use MyAdapter.

3) Adapt the build number 0..255 to return it at the *Version* method. The build number is a part of the version number which is completely user defined. The rest of the version number **must not be changed**, as it will be used for upwards/downwards compability checks of the engine.

4) Change the name and description at the *Capabilities* method

5) The SnowWhite adapter is using <asKey> methods for *ReadNext* / *Read* / *Insert* and *Update*, and for *Delete*. The example shows in a simple way how to do this operations with static elements. Replace them by your real database access.

6) Adapt the *Login* for different users. The SnowWhite login example just expects username= super and password=user (MD5 encoded) and returns the <sUserKey> = „5678“. If you're using the database adapter for the client side with only one user, you can implement it as dummy. Please note thate *CreateContext* is currently checking the returned <sUsrKey>

7) For the client engine you don't need an admin data implementation.
   For the server side, you can either
   - configure an ODBC implementation
   - use the INTERNAL_ADMIN implementation (using textdb way of using it)
   - implement your own admin part by overriding the virtual admin methods

8) For the BLOB implementation
   - use the INTERNAL_BLOB implementation (using textdb way of using it)
   - implement your own BLOB part by overriding the virtual BLOB methods

**NOTE:** Suspend/Resume is using the BLOB implementation for partial items, so running the datastore with OMA DS 1.2 requires a BLOB implementation.

9) Optionally implement now other things you need like filter support, e.g. for date ranges.

10) The SDK contains a lot of utiltiy functions ( SDK_util / SDK_support ) which can be used by the database adapter.

# 5. SySync UIApi SDK description

The SySync UIApi provides core SyncML functionality in form of a library (.dll, .so, .dlib etc. depending on the platform) to create SyncML applications not only with **custom specific UI** but also **custom specific SyncML communication layers**. This is because network communication is, especially on mobile devices, tightly coupled with the UI (asking user for network to use, connection to establish, certificates to accept or deny etc.). In addition, modern operating systems all provide built-in support libraries for common communication layers like HTTP or OBEX which match platform specifics optimally.

## 5.1 Connecting the SyncML core library via UIApi

The UI Api interface is based on an interface structure with several methods in it.

So as the first step, the UI application must get this **UI_Call_In** interface structure <aCI> and the engine's version number <aEngVersion> from the SyncML engine. There is a unified function call „ConnectEngine":

```
ENGINE_ENTRY TSyError ConnectEngine ( UI_Call_In *aCI,
                                      CVersion   *aEngVersion,
                                      CVersion    aPrgVersion,
                                      uInt16      aDebugFlags ) ENTRY_ATTR;
```

**NOTE:** For C# „**ConnectEngineS**" must be used instead. That's beause the interface structure must be allocated within the managed environment. <aCallbackVersion> is the current version of this structure, as it might increase for future versions. It is allowed to use „ConnectEngineS" (as replacement for „ConnectEngine") also in the C/C++ environment.

```
ENGINE_ENTRY TSyError ConnectEngineS( UI_Call_In  aCI,
                                      uInt16      aCallbackVersion,
                                      CVersion   *aEngVersion,
                                      CVersion    aPrgVersion,
                                      uInt16      aDebugFlags ) ENTRY_ATTR;
```

The UI_Call_In structure allows now to access to all the UI application functions (through its function pointer members). The UI_Call_In interface structure is based on the same **SDK_Interface_Structure** (defined at sync_dbapidef.h), which is also used by the DBApi SDK. The DBApi and UIApi share some of the functions – for example the DB_DebugXXXX functions can be called in both APIs for creating log file entries. Likewise, the GetValueXXX and SetValueXXX routines are available in both APIs.

For C++, there is a wrapper class named TEngineModuleBridge is provided as part of the SDK to facilitate access; likewise, for Borland/Codegear Delphi a similar Delphi wrapper class (Delphi\sdk_sources_delphi\sysync_engine.pas) in Pascal exists.

The engine version can be used for compatibility check or action. The engine itself makes some checks for compatibility with <aPrgVersion> versus <aEngVersion>, so is strongly recommended to take „Plugin_Version( <buildNumber> )" as <aEngVersion> or the version definitions of Java or C#.

The UI Api contains 3 sections:
- Engine Init
- Running Sync Sessions
- Settings Access

For detailed information see method descriptions in „enginemodulebase.h“

The UI application must disconnect the engine at the end to make sure that all settings are stored correctly:

/* Entry point for disconnecting the engine at the end */
```
ENGINE_ENTRY TSyError DisconnectEngine( UI_Call_In aCB ) ENTRY_ATTR;
```

## 5.2 Using a SyncML Client Library via UIApi

The following paragraphs describe the basic steps to take to create a SyncML client application. Details may differ depending on your actual setup.

Please also refer to the **fully functional sample clients** provided as part of the SDK: SyncML clients for Mozilla Sunbird/Lightning as GUI applications for Windows (in Codegear Delphi Pascal), MacOSX (in XCode Cocoa/Objective C) and the sample Contacts sync application for iPhoneOS.

Note that the SDK provides some glue code for the different platforms to simplify initialization and usage of the SyncML Client Library UIApi, and hide the binary API which is based on a C struct (UI_Call_In) containing function pointers.
For example, the iPhoneOS SDK contains Cocoa wrapper classes in the "XCode/sdk_sources_cocoa" directory, or the Windows SDK contains a Delphi wrapper class in "DELPHI/sdk_sources_delphi". For generic C++ access, the enginemodulebridge class in the "Sources" directory provides the UIApi as a C++ class.
These wrappers usually hide or abstract the ConnectEngine call described in 5.1 in a object, so connecting the engine consists of creating the object and passing the library path as a parameter. Please see the sample applications for details.

### *5.2.1 Preparation for initialisation*

First, the engine library must be connected as described in 5.1.

Then, before actually initializing the engine with a XML configuration file, some preparations might be needed. The Synthesis SyncML engine version 3.1 and later support so-called "config variables", which can be used to embed dynamic data from the runtime environment in a config file. For example, command line arguments can be used to define path specifications in the config file (debug logs, data files...) without actually modifying the config file.

Assume the config file contains a debug log directory specification as follows:

```
<debug>
  <logpath>$(mylogpath)</logpath>
  …
</debug>
```

To make this work, the config variable "mylogpath" must be defined before actually reading the XML config (see SySync_config_reference.pdf chapter "Configuration variables and conditional configuration"). In C, this will look as follows:

```
// Assume that UI_Call_In *callInP contains the call-in structure
// as returned by ConnectEngine()

TSyError sta;

// open the settings key that provides access to config variables
KeyH keyH = NULL; // will receive the opened key's handle
sta = callInP->OpenKeyByPath(callInP, &keyH, NULL, "/configvars",0);

if (sta==LOCERR_OK) {
  // config variable settings key opened successfully
  // - define text mode
  callInP->SetTextMode(callInP, keyH, CHS_UTF8, LEM_CSTR, false);
  // - set the config variable's value
  callInP->SetValue(
    callInP,
    keyH,
    "mylogpath",
    VALTYPE_TEXT,
    "C:\\syncml\\logfiles", // the config var value to set
    -1 // automatically calculate length from null-terminated string
  );
  // done with config variables, close settings key
  callInP->CloseKey(callInP,keyH);
  keyH=NULL;
}
```

When all config variables are defined, the engine can be initialized.

### 5.2.2 Engine Init

As the next step, the configuration must be provided to the SyncML engine. This can be done on three different ways. One of these ways must be chosen:

- the file name must be provided to the engine, so the engine can read the file directly (using **InitEngineFile**).
- the whole configuration must be presented as one contiguous data block in memory (using **InitEngineXML**).
- a callback for config reading must be given, so the engine can read it step by step (using **InitEngineCB**).

The string mode can be chosen (using **SetStringMode**) **prior** to the engine initialisation, if default settings are not fitting, e.g. a charset which is not UTF-8

### 5.2.3 Acessing Settings

There are several SyncML engine settings which can be configured from the UI application side. The settings are embedded within a tree, comparable to a directory or Windows registry tree. Setting values have therefore a path name, which must be opened first (with „**OpenKeyByPath**“). Then values can be read („**GetValue**“ / „**GetValueByID**“) or written ( „**SetValue**“ / „**SetValueByID**“ ). Multiple settings paths can be opened at the same time. „**SetTextMode**“ and „**SetTimeMode**“ can be defined for each context.

The UI Application interface does not provide any undo functionality, so changes will usually take effect immediately (or after calling „CloseKey“). If undo is required, the UI application itself must provide this functionality.

An example program „UI_app_example.cpp“ shows how these settings can be read and written and a simple sync session can be run. A list of common path and key names supported for the SyncML client engine can be found in 9.3. Depending on the version, special functionality and platform of the client library the available path and key names might be different from the standard set in 9.3 – please refer to separate manuals for specific products.

Examples for key paths are:
- „/profiles“
- „/engineinfo“

Examples for „/profile“ sub entries are:
- „serverURI“
- „serverPassword“

Example for „/engineinfo“ entries is:
- „version“

The policy should be to close keys immediately after use, this will avoid consistency and locking problems in some cases. Some values will be stored persistently, others must be set up each time the UI application is starting.

#### 5.2.3.1 Preparations before accessing settings profiles

At least one client settings profile should be present after initialisation. So it is recommended to check for an existing profile at startup of a client application and create a profile if none already exists. The following code sample shows the steps:

```
// Still assume that UI_Call_In *callInP contains the call-in structure
// as returned by ConnectEngine()

// access settings to make sure a profile exists
KeyH profilesKeyH=NULL, profileKeyH=NULL;
// - open the profiles container
sta = callInP->OpenKeyByPath(callInP,&profilesKeyH,NULL,"/profiles",0);
if (sta==LOCERR_OK) {
  // - first check settings status. This returns an error code if
  //   configuration data was found, but is not compatible with current
  //   version of the engine. If so, the "overwrite" flag must be
  //   explicitly set to force overwriting the old config with a new,
  //   empty copy.
  short settingsstatus;
  memSize sz;
   sta = callInP->GetValue(
    callInP,
```

```
      profilesKeyH,
      "settingsstatus",
      VALTYPE_INT16, // we want the value as 16-bit integer
      (appPointer)&settingsstatus, // put value here
      sizeof(settingsstatus), // size of variable
      &sz
    );
    if (sta!=LOCERR_OK || settingsstatus==LOCERR_CFGPARSE) {
      // problem with current config.
      // We could ask user here to preserve old config and
      // exit the application. For now, we just force
      // creation of a new config
      // - set "overwrite" flag to force creation of new config
      uInt8 overwrite=1;
      sta = callInP->SetValue(
        callInP,
        profilesKeyH,
        "overwrite",
        VALTYPE_INT8,
        &overwrite, // the config var value to set
        sizeof(overwrite) // size
      );
      // - now check status again (will create new settings in the engine)
      sta = callInP->GetValue(
        callInP,
        profilesKeyH,
        "settingsstatus",
        VALTYPE_INT16, // we want the value as 16-bit integer
        (appPointer)&settingsstatus, // put value here
        sizeof(settingsstatus), // size of variable
        &sz
      );
    }
    // see if at least one profile exists - if not, create default profile
    sta = callInP->OpenSubkey(
      callInP, &profileKeyH, profilesKeyH, KEYVAL_ID_FIRST, 0
    );
    if (sta==DB_NoContent) {
      // no profile exists, create default profile now
      sta = callInP->OpenSubkey(
        callInP, &profileKeyH, profilesKeyH, KEYVAL_ID_NEW_DEFAULT, 0
      );
      if (sta!=LOCERR_OK) {
        // Error, cannot create settings
        // You could show an user alert here
        exit(1); // terminate
      }
    }
    if (sta==LOCERR_OK && profileKeyH!=NULL) {
      // profile exists now
      callInP->CloseKey(callInP,profileKeyH); // close for now
    }
    // done with profiles for now
    callInP->CloseKey(callInP,profilesKeyH);
  }
```

### 5.2.3.2 Editing Settings

To provide editing of client settings, the applications must open the "/profiles" key as show above, then open one of the contained profiles. This profile contains some session-level configurati-

on like Server URL, username, password. As a session can *target* more than a single datastore for synchronisation, each profile contains a "targets" container which in turn contains a key for each datastore the client supports. To identify the targets, the XML configuration file must include a numeric identifier in the <dbtypeid> tag in each <datastore> section (see SySync_config_reference.pdf). Within the "targets" key, this identifier can be used to open the individual targets by id using OpenSubkey().
So the settings hierarchy for SyncML clients is as follows (details see 9.3):

- "/profiles" is the container of all client settings profiles. At least one profile is required, multiple profiles can be used to maintain settings for synchronizing with more than one SyncML server.
- "/profiles" contains the special "settingstatus" and "overwrite" values used to check "health" of current settings, as described in 5.2.3.1.
- Profiles within "/profiles" must be opened by using OpenSubkey(), usually by iterating over available profiles using the special KEYVAL_ID_FIRST and KEYVAL_ID_NEXT values as id.
    - Each profile contains a number of session level settings values, like "serverURI", "serverUser" etc. - these are accessed using GetValueXXX and SetValueXXX routines.
    - Each profile contains a "targets" container key which can be opened by OpenKeyByPath().
        - "targets" contains a target key for each datastore supported by the SyncML client engine (that is, those defined in the XML configuration).
        - Each target must be opened using OpenSubKey(), using the numeric identifier specified with <dbtypeid> in the XML config for each datastore. It is also possible to iterate over all targets using the special KEYVAL_ID_FIRST and KEYVAL_ID_NEXT values as id.
            - Each target contains a number of datastore level settings values, like "syncmode", "remotepath" etc. - these are accessed using GetValueXXX and SetValueXXX routines.

When accessing these settings, make sure you don't close container keys as long as subkeys contained are still open. So usually, keys are opened in the order "/profiles", profile, "targets", target and closed in the reverse order. It is allowed to have multiple profiles or targets open at the same time, as long as the parent key remains open as well.

### 5.2.4 Running Sync Sessions

Running a sync session consists of three basic steps:

- creating a sync session using OpenSession()
- calling SessionStep() repeatedly in a loop until it returns STEPCMD_DONE.
    - The return value in aStepCmd (see "engine_defs.h" for SESSIONSTEP_xxx definitions) must be checked to see when the engine has SyncML data ready to send to the SyncML server or needs an answer from the SyncML server. If so, the needed comminication with the server (http, OBEX) must take place using GetSyncMLBuffer()/RetSyncMLBuffer() or ReadSyncMLBuffer()/WriteSyncMLBuffer() routines to get or put SyncML data. For Java applications, only Read/WriteSyncMLBuffer are available.

  - To send data to the SyncML server, the application must query the SyncML engine for the URL and content type to use by opening the session-local session key using OpenSessionKey() and querying its "connectURI" and "contenttype" values.
  - The communication channel can be held open between calls to SessionStep() until STEPCMD_RESTART is returned in aStepCmd.
  - Each call to SessionStep() returns a record of TEngineProgressInfo type, which indicates progress of the sync session. The information in this record is useful to show progress in the UI of the application. See "engine_defs.h" for progress event PEV_XXX definitions.
- closing the sync session using CloseSession()

The following C code skeleton shows the basic implementation required to run a sync session:

```
// Assume that UI_Call_In *callInP contains the call-in structure
// as returned by ConnectEngine()

// run a sync session
// - variables
TEngineProgressInfo progressInfo;
SessionH sessionH = NULL;
TSyError sta;
uInt16 stepCmd = STEPCMD_CLIENTSTART; // first step
const memSize textbuffersize = 300;
      memSize textsize;
char textbuffer[textbuffersize];
// - create a session
sta = callInP->OpenSession(callInP,&sessionH,0,"mySyncSession");
if (sta!=LOCERR_OK) {
   // error, exit
   exit(1);
}
// sync main loop
do {
  // take next step
  sta = callInP->SessionStep(callInP,sessionH,&stepCmd,&progressInfo);
  if (sta!=LOCERR_OK) {
   // error, terminate with error
   stepCmd=STEPCMD_ERROR;
  }
  else {
   // step ran ok, evaluate step command
    switch (stepCmd) {
      case STEPCMD_OK:
        // no progress info, call step again
        stepCmd = STEPCMD_STEP;
        break;
      case STEPCMD_PROGRESS:
        // new progress info to show
        // Check special case of interactive display alert
        if (progressInfo.eventtype==PEV_DISPLAY100) {
          // alert 100 received from remote, message text is in
          // SessionKey's "displayalert" field
          KeyH sessionKeyH;
          sta = callInP->OpenSessionKey(callInP,sessionH,&sessionKeyH,0);
          if (sta==LOCERR_OK) {
            // get message from server to display
            callInP->GetValue(
              callInP,
              sessionKeyH,
              "displayalert",
              VALTYPE_TEXT, // we want the value as 16-bit integer
              (appPointer)&textbuffer, // put value here
```

```
            textbuffersize, // size of variable
            &textsize
          );
          // tbd: display message to user
          callInP->CloseKey(callInP,sessionH);
        }
      }
      else {
        // normal progress info
        // tbd: show progress in the UI
      }
      stepCmd = STEPCMD_STEP;
      break;
    case STEPCMD_ERROR:
      // error, terminate (should not happen, as status is
      // already checked above)
      break;
    case STEPCMD_RESTART:
      // make sure connection is closed and will be re-opened for next request
      // tbd: close communication channel if still open to make sure it is
      //      re-opened for the next request
      stepCmd = STEPCMD_STEP;
       break;
    case STEPCMD_SENDDATA:
      // send data to remote

      // tbd: use OpenSessionKey() and GetValue() to retrieve "connectURI"
      //      and "contenttype" to be used to send data to the server
      // tbd: use GetSyncMLBuffer()/RetSyncMLBuffer() to access the data to be
      //      sent or have it copied into caller's buffer using
      //      ReadSyncMLBuffer(), then send it to the server

      // status for next step
      if (true) /* tbd: check if communication with server successful */
        stepCmd = STEPCMD_SENTDATA; // we have sent the request data
      else
        stepCmd = STEPCMD_TRANSPFAIL; // communication with server failed
       break;
    case STEPCMD_NEEDDATA:
      // tbd: wait for receiving answer from server

      // tbd: put answer received into SyncML engine's buffer, either by
      //      directly accessing it using GetSyncMLBuffer()/RetSyncMLBuffer()
      //      or by copying it with WriteSyncMLBuffer().

      // status for next step
      if (true) /* tbd: check if communication with server successful */
        stepCmd = STEPCMD_GOTDATA; // we have received response data
      else
        stepCmd = STEPCMD_TRANSPFAIL; // communication with server failed
      break;
   } // switch stepcmd
  }
  // check for suspend or abort, if so, modify step command for next step
  if (false /* tdb: check if user requests suspending the session */) {
    stepCmd = STEPCMD_SUSPEND;
   }
  if (false /* tdb: check if user requests aborting the session */) {
    stepCmd = STEPCMD_ABORT;
  }
  // loop until session done or aborted with error
} while (stepCmd!=STEPCMD_DONE && stepCmd!=STEPCMD_ERROR);
// done, now close the SyncML session
sta = callInP->CloseSession(callInP,sessionH);
```

SessionStep() is designed to keep execution time as short as possible, such that implementing a responsive SyncML client is possible without using a separate thread. The code skeleton above

can be integrated in a GUI application main loop to allow processing SyncML, showing progress in the UI and responding to user's requests (like pressing an abort button) in parallel.

**NOTE:** It's recommend (on Windows systems) to initialize the network access before opening the session for the first time. That's because the <LocURI> information will be taken from network information, if available.

An example program, **UI_app_example** is part of the SDK package for most of the supported platforms. It shows how to run a simple SyncML session. This example uses the open source package **CURL** as network interface (which is not part of the SDK package). The sources are requested at the prepared (but empty) **libcurl** directory.

For **Android**, an example program with a complete Android user interface of the uiapp.java example is provided.

# 6. Setup Guide

This setup guide consists of two main sections:

- *Section 6.1* describes the installation of the **C/C++** interface,
  *Section 6.3* describes the usage of the JNI interface for **Java**.
  *Section 6.3* describes the usage of the interface for **C#**.

- We recommend, that if you are new to Synthesis SyncML Server/Client, start testing with the standalone version as you have immediate visible feedback on the console screen. If everything is working fine, then you can easily switch to the ISAPI or Apache version. For installation of the ISAPI or Apache version see the description of „SySync_Server_manual".

## 6.1 Plug-in System for C/C++

**Metrowerks CodeWarrior** project files ( .mcp) are part of the SDK delivery for **Windows**, as well as the compiled shared libraries. The examples are based on CodeWarrior V8.2 for Windows.

For **Windows** alternatively **Visual Studio** 2005 or newer can be used. Ready-to-use *.vcproj files are part of the SDK package.

For **Linux**, a generated makefile „Sysync_SDK_linux.mk" is part of delivery as well and can be used directly by calling „**make**": make –f Sysync_SDK_linux.mk.

For **MacOSX** **XCode** can be used to create Universal Binaries which are working on PPC and X86 architectures.

Three plug-in modules (a simple demo module in pure „C" , a text DB module in „C++" and a extendable C++ adapter „snowwhite") can be compiled and linked directly. Result will be the three shared library modules „SDK_demodb" , „SDK_textdb" and „snowwhite".

| | |
|---|---|
| .dll | for Windows, |
| .so | for Linux, |
| .dylib | for MacOSX |

- Standard C example: The „SDK_demodb" is just printing a debug message for each routine. This can be a good and helpful starting point to implement routine by routine. Writing debug messages is done thru the callback mechanism of the SyncML engine, using the „Debug_DB" call. The debug messages will be stored in the SyncML engine's log files. Don't use „printf" calls, as not all versions of the Synthesis SyncML server are able to create such kind of output.

- C++ example: The „SDK_textdb" is a text DB interface, which acts the same way as the so called Synthesis SyncML demo server. This module can be a starting point when it's easier to adapt from an already running system. Plugin parameters <datafilepath>, <blobfilepath> and <mapfilepath> are supported.

**NOTE:** The CurrentTime() function used at „Session_GetDBtime“ and „StartDataRead“ is implemented very rudimentary. For a good implementation, this function should be replaced by an enhanced version.

The example configuration „syncserv_odbc.xml“ is set up to access the „SDK_textdb“ directly by default. <plugin_module> is configured for „SDK_textdb“ on session and datastore level. The admin tables of the „SDK_textdb“ will be used.
For Linux and MacOSX „LD_LIBRARY_PATH“ must be set in order to access these plugin library modules.

## 6.2 Plug-in System for the iPhone

The iPhone apps do not allow the usage of Java nor of dynamically linked libraries. So the database plugins must be **statically linked** with predefined names.
Additionally Cocoa does not support the C++ namespaces which are used for built-in plugins.

The solution which has been chosen are **4 predefined built-in bridge modules "[iPhone_dblugin1]" .. "[iPhone_dblugin4]"**. They are connected via sample_dbpluginX_wrapper.mm to the Cocoa „**sample_dbplugin**“ (.h and .m). They can be changed to the user's SDK Cocoa plugin module best by making a copy of sample_dbplugin as starting point and implementing there the system specific readnext/read/insert/update and delete functionality.

## 6.3 Plug-in System for Java

The Java Virtual Machine V1.4 or higher must be correctly installed.
For Linux and MacOSX „LD_LIBRARY_PATH“ must be set to access the JavaVM, e.g. at “/usr/java/jre/lib/i386/client“ or „/usr/java/jre/lib/i386/server“.
For **MacOSX** this is normally „/system/Library/Frameworks/JavaVM.framework“. By default it's searching for **libjvm.dylib**. For SDK 1.9.0 and higher it also searches for **JavaVM.dylib**.

For **Android** everything in Java is set up with the „uiapp“ sample as eclipse project. At least Android SDK **1.6** is requested to run correctly with the NDK library „libsynthesis_client.so“ Synthesis recommends to compile with Android **2.0**, but allow Android **1.5** as min version.

There is a Java example „SDK_javadb.java“ and its compiled classes (which have been created with „javac SDK_javadb.java“).
The SDK class is within a **Java package**: A package example (named **com.sysync**) can be found in the according subdirectory. A package can be accessed by adding the package path or by adding the package name, separated by a space. **JavaVM options** can be added **after** the class and package name.

| | | |
|---|---|---|
| **Examples:** | [JNI]**!**/com/sysync/SDK_javadb | (using com.sysync package) |
| | [JNI]**!**SDK_javadb com/sysync | (alternative syntax) |
| | [JNI]**!**/com/sysync/SDK_javadb –verbose:jni | (with additional options) |
| | [logger**!**JNI]**!**/com/sysync/SDK_javadb | (using additional logger) |
| | [JNI]!/com/sysync/SDK_javadb –Xrs | (^C will not be disabled) |
| | [JNI]!/com/sysync/SDK_javadb -Djava.class.path=/xxx | (with a different classpath) |

The SyncML engine (**PRO** version only) has a built-in plug-in module for the JNI handling.

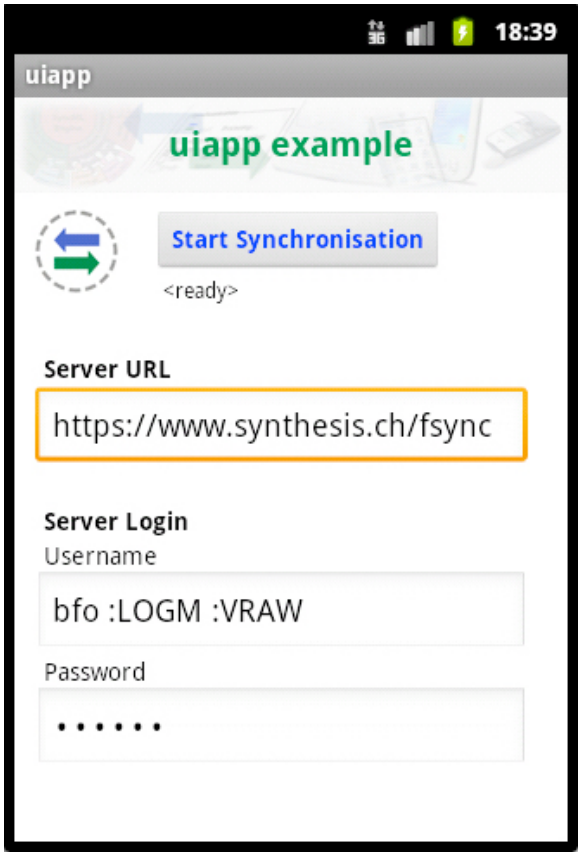
In earlier SDK versions, Java sources outside any package and within „sysync“ package have been provided. The entry points are still valid, also for the 32-bit interface (for contexts, pointers and sizes), but it is recommended to switch over to the com.sysync package version with the 64 bit interface.

For **UI applications**, written in Java, the engine connection will be done with the „Sysync_ConnectEngine“ call, which then provides access to all the UI application specific functions. A simple sample program how to use the interface is part of the SDK: **uiapp_main.java**. NOTE: For some systems the architecture must be defined with –d32 or –d64

**Examples:** java –cp . com.sysync.uiapp_main
java –d64 –cp . com/sysync/uiapp_main

### 6.3.1 Android setup

- Android's SDK (at least **1.6** to support NDK 1.6; Android SDK **2.2** is recommended, which supports also **2.3** und **3.0/3.1** apps, min version **1.5** is possible) and Eclipse must be installed.
- The **uiapp example** project can be directly installed as Android app (browse for the path where the AndroidManifest.xml is located).
- A valid license code (temporary or permanent) is required, the existing „res/raw/license.txt“ file must be replaced (1st line license text, 2nd line license code).




The uiapp example is able to run a complete SyncML session, the example datastores are however only rudimentary and must be completed.

## 6.4 Plug-in System for C#

Will be accessed via GUID. There is a common GUID for the interface (which shouldn't be changed and a class specific GUID. The class specific GUID will be expected as subname at the config file. GUIDs must be registered to the system using „regasm".

**regasm dbapi_csharp.dll /tlb: dbapi_csharp.tlb**

**Example:**
<plugin_module>CSHARP!348330a6-c7ee-4ba4-888b-39250cb31db1"</plugin_module>

## 6.5 Plug-in module XML configuration

Plug-in modules must be activated at the XML configuration file.
This can be done for the session and/or each datastore separately. It is even possible to use different plug-ins for the session and each datastore.

For the **session**, <server type="plugin"> must be set.
The <plugin_module> must be set to the plug-in name, e.g.
<plugin_module>SDK_textdb</plugin_module>

As there are built-in modules and external modules, the syntax for internal plug-ins is using brackets. Example:
- Internal demo plug-in: <plugin_module>[SDK_demodb]</plugin_module>
- Shared library for demo plug-in: <plugin_module>SDK_demodb</plugin_module>

Built-in means, that the plug-in is compiled and statically linked to the SyncML engine.
Only Synthesis can create built-in modules; the SDK user can only create external modules as shared libraries. „SDK_demodb" and „SDK_textdb" are available both as built-in and as external modules. The **PRO** version of the Synthesis server contains the JNI as built-in plug-in module.
For system performance, there is no measurable difference between internal / external modules.

By default, the SyncML engine doesn't use authentification and device administration of the plug-in module. They must be switched on in the configuration.
To use the login part of the plug-in, <plugin_sessionauth>yes</plugin_sessionauth> must be set. To use the admin part of the plug-in, <plugin_deviceadmin>yes</plugin_deviceadmin> is required. Some SyncML engines are always using the internal binfile system for admin, on these the activation of plugin_deviceadmin has no effect.
For the **datastore** access, <datastore name="NAME" type="plugin"> must be set, where the NAME is e.g. „contacts". The <plugin_module> must be set the same way as for the session.
If <plugin_deviceadmin> is active on session level, it will be used for each datastore as well.

## 6.6 Module naming convention

The Synthesis SyncML plug-in modules will used with the following naming convention:

<plugin_module>name</plugin_module>

A shared library „name.dll“ ( or „name.so“ for Linux or „name.dylib“ for MacOSX) must exist in the search path of the system. The SyncML engine will create an error 20010, if the library does not exist. „name“ can consist of a main module name and sub module names, e.g. „aaa!bbb“, where „aaa“ is the name of the library ( „aaa.dll“ ) and ‚bbb’ is a local name for the plug-in module. This concept allows to build cascades or hierarchies of plug-ins, where the sub module name is given to the next plug-in module. „aaa!bbb!ccc“ is an example for a plug-in with a two level sub system.

Example: The bridge to Java
The Java plug-in is an example of such a sub system: The syntax for the SDK sample must be:

<plugin_module>JNI!SDK_javadb</plugin_module>

Here we have a plug-in called „JNI.dll“ which is the bridge between the SyncML engine (written in C++) and the the Java plug-in (SDK_javadb) with the static class „SDK_javadb“.
Error 20034 will be returned, if the sub system is not available.

For Java, the **CLASSPATH** can be included directly as well.
As an example for a classpath at „C:\java_files“, the Java Virtual Machine will be attached correctly with:

<plugin_module>[JNI!C:\java_files\SDK_javadb]</plugin_module>

**NOTE:** SDK V1.3.8.X and later allow to connect **more than one Java class** with a given classpath and package name for different plugin sections, however the VM options are the same and will be shared among all Java plugins. The options of the first connecting context (usually the session context) will considered, options of other <plugin_module> sections will be ignored.

**NOTE:** The plugin module names are **case sensitive** in some cases, e.g. in Java environment.

**C# database adapters** are using a GUID as sub system name:

Example: C# bridge

<plugin_module>CSHARP!348330a6-c7ee-4ba4-888b-39250cb31db1"</plugin_module>

## 6.7 Plugin_Info program

A utility program for testing plugin modules independently is part of the SDK. It returns some version and identification info about a plugin:

Example: `plugin_info –p SDK_textdb`

```
Synthesis Plugin_Info Tool V1.9.2.0

Name         : SDK_textdb
Version      : V1.9.2.0
... at least : V1.5.1.0 required as engine's SDK version

Version      : V1.7.0.0
Manufacturer : Synthesis AG
Description  : Text database module. Writes data directly to TDB_*.txt file
Platform     : Windows (CodeWarrior) (DLL)
GlobContext  : 0014AAB0 (00159868 'TextDB')

plugin_sessionauth    : yes
plugin_deviceadmin    : yes
plugin_datastoreadmin : yes
plugin_datastore      : yes
plugin_dataadapt      : yes
ADMIN_Info            : yes

Session   context         :     available (0015C1F0)
Datastore context (admin) :     available (0015C280)
Datastore context (data)  :     available (0015CF58)
```

Example: `plugin_info –p '[JNI]!/com/sysync/SDK_javadb -Xrs'`

```
Synthesis Plugin_Info Tool V1.7.0.0

Name         : [JNI!/com/sysync/SDK_javadb]
Version      : V1.9.2.0
... at least : V1.6.1.0 required as engine's SDK version

Name         : JNI
Version      : V1.9.2.0
Manufacturer : Synthesis AG
Description  : JNI bridge to Java
Platform     : Windows (CodeWarrior) (LIB)
GlobContext  : 0014B130 (00145158 'JNI')

sub system
----------
Name         : /com/sysync/SDK_javadb
Arguments    : -Xrs
Version      : V1.9.2.0
Manufacturer : Synthesis AG
Description  : Java Example Module (com.sysync)
Platform     : Java


plugin_sessionauth    : yes
plugin_deviceadmin    : yes
plugin_datastoreadmin : yes
plugin_datastore      : yes
plugin_dataadapt      : yes
ADMIN_Info            : yes

Session   context         :     available (0014C1B8)
Datastore context (admin) :     available (0014C430)
Datastore context (data)  :     available (0014C6B8)
```

The program allows to get some **debug information** (with options –d and –e) as well as forcing any desired **engine SDK version** other than the current version for test (option –v).

For Linux and MacOSX „LD_LIBRARY_PATH“ must be set in order to access these plugin library modules.

## 6.8 UIApi C# interface

For C# there exists an equivalent SDK interface to the UI functions of the engine. The usage of the functions is identical, they are implemented via delegate marshal calls.
The interface definition can be found in the program.cs example of the SDK package.
„ConnectEngineS“ and „DisconnectEngine“ are implemented as DLL calls.

The difference to the function „ConnectEngine“ is, that the SDK_Interface_Struct will allocated on the C# side; <aCallbackVersion> will be needed additionally.

```
class uiapp
{
  // ...

 // ------------------------------------------------------------------------------
  [DllImport(DllName)] public static extern
  TSyError ConnectEngineS  ( ref SDK_Interface_Struct aCB,
                                          UInt16 aCallbackVersion,
                                    out CVersion aEngVersion,
                                        CVersion aPrgVersion,
                                          UInt16 aDebugFlags );
  [DllImport(DllName)] public static extern
  TSyError DisconnectEngine( ref SDK_Interface_Struct aCB );

  // ...
}
```

As an example, the „InitEngineXML“ call is shown below. The whole UI call in is implemented as class „uiapp_calls“ at „sysync_uiapp_calls“ as part of the SDK package.

```
public class uiapp_calls
{
  // the Call-In structure
  public SDK_Interface_Struct fCB = new SDK_Interface_Struct();

  // The method prototype definitions ...

  delegate TSyError InitEngineXML_Func ( ref SDK_Interface_Struct aCB, String aConfigXML );

  // ...

  // ... and the marshal calls
  public TSyError InitEngineXML( String aConfigXML )
  {
    IntPtr ip= fCB.InitEngineXML;
                  InitEngineXML_Func v=
                 (InitEngineXML_Func)Marshal.GetDelegateForFunctionPointer( ip,
           typeof(InitEngineXML_Func) );

    return v( ref fCB, aConfigXML );
  } // InitEngineXML

  // ...
}
```

# 7. Change History

## 7.1 Changes for SDK 1.3.0

For customers who have started to implement a plug-in with SDK V1.0.0.2, some adaptions must be made to make it conformant to SDK V1.7.0. Best way to do this, is to install this new SDK package and adapt the plug-in's c/cpp or java files. The SyncML engine is written to be **downwards** compatible for all these features, so it will run correctly with all versions, as long as the „Plugin_Version" of SDK_util will return the correct version number, so this version number MUST NOT be changed ( except for the customer defined build number, which represents the last digit 0..255 of the version number ).

**Upwards** compatibility ( SDK version newer than engine ) are covered partly, because an older engine cannot handle cases of changed parameter lists or no longer existing functions or methods. However the engine informs the plugin about its version (NOTE: this is not SyncML server's / client's version, e.g. V2.1.1.28, but the plugin interface version, e.g. V1.7.0.0). Additionally the plugin returns (with „Module_Capabilities" ) the minimum required version. The engine automatically stops then with an error **20033** if the this build is too old for this SDK/plugin.

The following things **MUST BE CHANGED**:

| | |
|---|---|
| **Module_PluginParams** | Has now a new additional parameter <engineVersion> (to inform the plug-in about its version) |
| Callback calls | Up to **callbackVersion = 5** is now supported, see structure definition at „sync_dbapidef.h". Hierarchical logs and exotic output are supported now. Boolean alignment problems are fixed. |
| **DeleteBlob** | New function added ( allows consistent BLOB handling now ) |
| **Session_AdaptItem, AdaptItem** | New functions added, these function will allow the manipulation of variables of the script context ( They are not yet supported by the SyncML engine, but the implementation is prepared now ). |
| **InsertMapItem** | New function added, no longer combined with „UpdateMapItem" |
| **MapID** | The MapID struct contains a new element **uInt8 ident**.. It will be used to store the status of Suspend/Resume of OMA DS 1.2. |
| Build numbers | System has changed from V1.0.N.2, where „N" was the platform identifier, to V1.3.8.X, where „X" is a customer defined build number. |
| **StartDataRead** | Has now a new additional parameter <resumeToken> for OMA DS 1.2 support ( Suspend/Resume ). |
| **Session_GetDBTime** | New function added ( returns the database's time, if available ) |
| **DB_Full** | Definition has been changed from 520 to **420**. |
| DBG_PLUGIN_XXX | A separate flag system (with 16 flags) for the plugin debug logging has been introduced. Bits 0 and 1 are defined and reserved. |

| | |
|---|---|
| DEBUG_Exotic | Has been separated into **DEBUG_Exotic_Call** and **DEBUG_Exotic_DB**. |
| **Session_Login** (Java only) | Parameter „String sPwd“ has been changed to „VAR_String sPwd“, to allow the usage of all password modes. |

## 7.2 Changes for SDK V1.4.0

| | |
|---|---|
| **JCallback** (Java only): | The callback functions are now using <thisCB> as 1st return parameter. This solves a problem with using Java packages.<br><br>**Example:**<br>`public native void DebugDB( int thisCB, String aTxt );` |
| **Module_Capabilites** | „ADMIN_Info“ will be considered now, it adds the identifier „ADMIN“ to <moduleName> at admin's „CreateContext“. |
| **FilterContext** | Is supported by the SyncML engine for SDK version >= V1.3.8. |
| Callback calls | Up to **callbackVersion = 8** is now supported, see structure definition at „sync_dbapidef.h“. |
| UI Api | Is supported by the SyncML engine for SDK version >= V1.4.0 **callbackVersion = 8** with the UI interface functions will be used. |
| DatastoreContext | 4 additional functions „***AsKey**“ have been added. They allow an <aItemKey> handling instead of using <aItemData>. Is supported by the SyncML engine for SDK version >= V1.4.0 |
| **SDK_Interface_Structure** | is the new name of the former „DB_Callback_Structure“. Two (**equivalent**) pointer types for this structure are defined now: The already existing „**DB_Callback**“ for the DBApi SDK and „**UI_Call_In**“ for the UIApi SDK. |

## 7.3 Changes for SDK V1.5.0

| | |
|---|---|
| C# (dbapi) | DebugDB and other callback calls can log now to the engine's log file directly: „sysync_debug_calls.cs“ is part of the SDK now.<br><br>[MarshalAs(UnmanagedType.LPWStr)] will be used now for all string parameters (instead of LPStr), the SyncML engine will make the conversion from/to UTF-8 internally.<br>The new files „dbapi_interface.cs“ and „sysync_debug_calls.cs“ (with the LPWStr defs) must be used. |
| C# (uiapp) | The UI call in methods have changed from direct DLL calls to marshalled delegate calls: „sysync_uiapp_calls.cs“ is part of the SDK now. |
| C# **ConnectEngineS** | Must be used instead of „ConnectEngine“ now. |

| | |
|---|---|
| InsertItem | DB Plugin may return a DB_DataMerged (207) error which informs the engine about an already existing item which has been updated now. The engine will read this item again to keep track of this update (either in the same or the next session).<br>**NOTE:** This feature is supported for server only. |
| Callback calls | Up to **callbackVersion = 9** is now supported, see structure definition at „sync_dbapidef.h“. |
| **FinalizeLocalID** | New function has been added to replace a temporary ID sion by its final ID at the end of the session. The function can return LOCERR_NOTIMP, if not needed. |
| **DeleteSyncSet** | All elements of the sync set will be removed. If the function returns LOCERR_NOTIMP, the engine will remove element by element. |

## 7.4 Changes for SDK V1.6.0

| | |
|---|---|
| GlobContext | „**GlobContext**“ extended for multi module usage |
| **SDK_textdb** example | Adapted for GlobContext use /<br>NID_* file has been added, containing a persistent new ID for inserted items. |
| Callback calls | Up to **callbackVersion = 11** is now supported, see structure definition at „sync_dbapidef.h“. |
| **OceanBlue / SnowWhite** | C++ example added |
| **SessionH / KeyH** | Colored pointer types introduced, binary compatible to appPointer |
| **UI_app_example** | Command line contacts sync; replaces the UI_app_setting |
| 64-bit support | Plugins can be compiled for 32 bit and 64 bit architectures. |
| Tunnel adapter | The tunnel interface is now supported by the engine |
| **Android** | Support for Android added, using com.sysync Java package |

## 7.5 Changes for SDK V1.6.2

| | |
|---|---|
| **Android** | Support for Android NDK 1.6 |
| Java | **com.sysync** package and 64 bit **long** signatures used<br>(older entry points still available for compatibility reasons) |
| Java | Entry points for server engine added |
| MacOSX | CodeWarrior no longer supported, please use XCode |

## 7.6 Changes for SDK V1.7.0

| | |
|---|---|
| **Android** | Support for up to **Android 2.1**<br>Use of Synthesis SyncML Engine 3.4.0.6<br>Synthesis Android client 1.7.0.25 is based on this SDK version |
| Java | Session default plugin added<br>Several utility helpers added |
| /engineinfo | Several **custom** values can be set now |

## 7.7 Changes for SDK V1.8.0

| | |
|---|---|
| iOS | iOS 4.0 support for iPhone/iPad |

## 7.8 Changes for SDK V1.9.0

| | |
|---|---|
| **Android** | Support for up to **Android 2.3**<br>Use of Synthesis SyncML Engine 3.4.0.21<br>Synthesis Android client 1.9.5 is based on this SDK version |
| Java | MacOSX Java is searching as well for JavaVM.dylib. |

## 7.9 Changes for SDK V1.9.1

| | |
|---|---|
| **Android** | Use of Synthesis SyncML Engine 3.4.0.24 |
| iOS | multiple user plugin support for iPhone/iPad |
| Java | enhanced **SDK_javadb** example available with AsKey example |

## 7.10 Changes for SDK V1.9.2

- Using the Synthesis SyncML Engine 3.4.0.30
- MacOSX is no longer statically linked with Java Framework

For more details of the SDK change history see „sync_dbapidef.h“.

# 8. DBApi Interface description

## 8.1 Function overview

- **TSyError Module_CreateContext** (**CContext** *mContext, **cAppCharP** moduleName, **cAppCharP** subName, **cAppCharP** mContextName, **DB_Callback** mCB)
- **CVersion Module_Version** (**CContext** mContext)
- **TSyError Module_Capabilities** (**CContext** mContext, **appCharP** *mCapabilities)
- **TSyError Module_PluginParams** (**CContext** mContext, **cAppCharP** mConfigParams, **CVersion** engineVersion)
- void **Module_DisposeObj** (**CContext** mContext, void *memory)
- **TSyError Module_DeleteContext** (**CContext** mContext)

- **TSyError Session_CreateContext** (**CContext** *sContext, **cAppCharP** sessionName, **DB_Callback** sCB)
- **TSyError Session_AdaptItem** (**CContext** sContext, **appCharP** *sItemData1, **appCharP** *sItemData2, **appCharP** *sLocalVars, **uInt32** sIdentifier)
- **TSyError Session_CheckDevice** (**CContext** sContext, **cAppCharP** aDeviceID, **appCharP** *sDevKey, **appCharP** *nonce)
- **TSyError Session_GetNonce** (**CContext** sContext, **appCharP** *nonce)
- **TSyError Session_SaveNonce** (**CContext** sContext, **cAppCharP** nonce)
- **TSyError Session_SaveDeviceInfo** (**CContext** sContext, **cAppCharP** aDeviceInfo)
- **TSyError Session_GetDBTime** (**CContext** sContext, **appCharP** *currentDBTime)
- **sInt32 Session_PasswordMode** (**CContext** sContext)
- **TSyError Session_Login** (**CContext** sContext, **cAppCharP** sUsername, **appCharP** *sPassword, **appCharP** *sUsrKey)
- **TSyError Session_Logout** (**CContext** sContext)
- void **Session_DisposeObj** (**CContext** sContext, void *memory)
- void **Session_ThreadMayChangeNow** (**CContext** sContext)
- void **Session_DispItems** (**CContext** sContext, **bool** allFields, **cAppCharP** specificItem)
- **TSyError Session_DeleteContext** (**CContext** sContext)

- **TSyError CreateContext** (**CContext** *aContext, **cAppCharP** aContextName, **DB_Callback** aCB, **cAppCharP** sDevKey, **cAppCharP** sUsrKey)
- **uInt32 ContextSupport** (**CContext** aContext, **cAppCharP** aContextRules)
- **uInt32 FilterSupport** (**CContext** aContext, **cAppCharP** aFilterRules)
- **TSyError LoadAdminData** (**CContext** aContext, **cAppCharP** aLocalDB, **cAppCharP** aRemoteDB, **appCharP** *adminData)
- **TSyError SaveAdminData** (**CContext** aContext, **cAppCharP** adminData)
- **bool ReadNextMapItem** (**CContext** aContext, **MapID** mID, **bool** aFirst)
- **TSyError InsertMapItem** (**CContext** aContext, **cMapID** mID)
- **TSyError UpdateMapItem** (**CContext** aContext, **cMapID** mID)
- **TSyError DeleteMapItem** (**CContext** aContext, **cMapID** mID)
- void **DisposeObj** (**CContext** aContext, void *memory)
- void **ThreadMayChangeNow** (**CContext** aContext)
- void **WriteLogData** (**CContext** aContext, **cAppCharP** logData)

- void **DispItems** (**CContext** aContext, **bool** allFields, **cAppCharP** specificItem)
- **TSyError AdaptItem** (**CContext** aContext, **appCharP** *aItemData1, **appCharP** *aItemData2, **appCharP** *aLocalVars, **uInt32** aIdentifier)
- **TSyError StartDataRead** (**CContext** aContext, **cAppCharP** lastToken, **cAppCharP** resumeToken)
- **TSyError ReadNextItem** (**CContext** aContext, **ItemID** aID, **appCharP** *aItemData, **sInt32** *aStatus, **bool** aFirst)
- **TSyError ReadNextItemAsKey** (**CContext** aContext, **ItemID** aID, **KeyH** aItemKey, **sInt32** *aStatus, **bool** aFirst)
- **TSyError ReadItem** (**CContext** aContext, **cItemID** aID, **appCharP** *aItemData)
- **TSyError ReadItemAsKey** (**CContext** aContext, **cItemID** aID, **KeyH** aItemKey)
- **TSyError ReadBlob** (**CContext** aContext, **cItemID** aID, **cAppCharP** aBlobID, **appPointer** *aBlkPtr, **memSize***aBlkSize, **memSize** *aTotSize, **bool** aFirst, **bool** *aLast)
- **TSyError EndDataRead** (**CContext** aContext)
- **TSyError StartDataWrite** (**CContext** aContext)
- **TSyError InsertItem** (**CContext** aContext, **cAppCharP** aItemData, **ItemID** newID)
- **TSyError InsertItemAsKey** (**CContext** aContext, **KeyH** aItemKey, **ItemID** newID)
- **TSyError FinalizeLocalID** (**CContext** aContext, const **ItemID** aID, **ItemID** updID)
- **TSyError UpdateItem** (**CContext** aContext, **cAppCharP** aItemData, **cItemID** aID, **ItemID** updID)
- **TSyError UpdateItemAsKey** (**CContext** aContext, **KeyH** aItemKey, **cItemID** aID, **ItemID** updID)
- **TSyError MoveItem** (**CContext** aContext, **cItemID** aID, **cAppCharP** newParentID)
- **TSyError DeleteItem** (**CContext** aContext, **cItemID** aID)
- **TSyError DeleteSyncSet** (**CContext** aContext)
- **TSyError WriteBlob** (**CContext** aContext, **cItemID** aID, **cAppCharP** aBlobID, **appPointer** aBlkPtr, **memSize** aBlkSize, **memSize** aTotSize, **bool** aFirst, **bool** aLast)
- **TSyError DeleteBlob** (**CContext** aContext, **cItemID** aID, **cAppCharP** aBlobID)
- **TSyError EndDataWrite** (**CContext** aContext, **bool** success, **appCharP** *newToken)
- **TSyError DeleteContext** (**CContext** aContext)

## 8.2 Function Documentation

**_ENTRY_ TSyError AdaptItem (CContext *aContext*, appCharP * *aItemData1*, appCharP * *aItemData2*, appCharP * *aLocalVars*, uInt32 *aIdentifier*)**

This function adapts aItemData

**Parameters:**

<*aContext*> The datastore context
<*aItemData1*> The 1st item's data
<*aItemData2*> The 2nd item's data
<*aLocalVars*> The local vars
<*aIdentifier*> To identify, where it is called

**Returns:**

error code

NOTE: The memory for adapted strings must be allocated locally. The SyncML engine will call 'DisposeObj' later, to release again its memory. One or more strings can be returned unchanged as well.

### _ENTRY_ uInt32 ContextSupport (CContext *aContext*, cAppCharP *aSupportRules*)

This function asks for specific context configurations

**Parameters:**

<*aContext*> The datastore context

<*aSupportRules*> The SyncML sends a list of support rules. This function has to reply, up to which rule, contexts are supported (and switched on now). Data is formatted as multiline aa:bb<CRLF>cc:dd[<CRLF>]

**Returns:**

Up to <n> fields are supported (and switched on) for this context. If 0 will be returned, no field of <aSupportRules> is supported.

### _ENTRY_ TSyError CreateContext (CContext * *aContext*, cAppCharP *aContextName*, DB_Callback *aCB*, cAppCharP *sDevKey*, cAppCharP *sUsrKey*)

This routine is called to create a new context for a datastore access. It must allocate all resources for this context and initialize the <aContext> parameter with a value that allows re-identifying the context. <aContext> can either be a pointer to the local context structure or any key value which allows to re-identify the context later. Subsequent calls related to this context will pass the <aContext> value as returned from CreateContext. The context must be valid until 'DeleteContext' is called. <plugin_params> can be defined individually.

NOTE: The SyncML engine treats <aContext> simply as a key. The only condition is uniqueness for all datastore contexts. Even <aContext> = 0 can be used.

**Parameters:**

<*aContext*> Returns a value, which allows to identify this datastore context.

<*aContextName*> Allows to identify the context, if more than one must be handled. <contextName> is defined at the XML configuration.

<*aCB*> **DB_Callback** structure for datatstore logging.

<*sDevKey*> The result of 'Session_CheckDevice' comes in here.

<*sUsrKey*> The result of 'Session_Login' comes in here.

**Returns:**

error code, if context could not be created (e.g. not enough memory), 0 if context successfully created.

### _ENTRY_ TSyError DeleteBlob (CContext *aContext*, cItemID *aID*, cAppCharP *aBlobID*)

This routine deletes the specific binary logic block <blobID> at the database.

**Parameters:**

<*aContext*> The datastore context.

<*aID*> **ItemID** ( with <item>,<parent> ).

<*aBlobID*> The assigned ID of the blob.

**Returns:**

error code, if not ok ( e.g. invalid <aID>,<aBlobID> )

### _ENTRY_ TSyError DeleteContext (CContext *aContext*)

This routine is called to delete a context, that was previously created with 'CreateContext'. The DB Module must free all resources related to this context. No calls with <aContext> will be done after calling this routine, so the assigned structure, allocated at 'CreateContext' can be released here.

**Parameters:**

<*aContext*> The datastore context.

**Returns:**

error code, if context could not be deleted ( e.g. not existing <aContext> ).

### _ENTRY_ TSyError DeleteItem (CContext *aContext*, cItemID *aID*)

This routine deletes a dataset from the database

**Parameters:**

<*aContext*> The datastore context.

<*aID*> **ItemID** ( with <item>,<parent> ) to be deleted.

**Returns:**

error code

- LOCERR_OK ( =0 ), if successful
- DB_NotFound ( =404 ), if unknown <aItemID>
- ... or any other SyncML error code, see Reference Manual

## _ENTRY_ TSyError DeleteMapItem (CContext *aContext*, cMapID *mID*)

Map table handling: Delete a map item of this context

**Parameters:**

<*aContext*> The datastore context

<*mID*> **MapID** ( with <localID>,<remoteID> and <flags> ).

**Returns:**

error code, if this **MapID** can't be deleted, or if this **MapID** does not exist.

USED ONLY WITH <plugin_datastoredadmin>

## _ENTRY_ TSyError DeleteSyncSet (CContext *aContext*)

This routine deletes all datasets from the database

**Parameters:**

<*aContext*> The datastore context.

**Returns:**

error code

- LOCERR_OK ( =0 ), if successful
- LOCERR_NOTIMP ( =20030 ). For this case, the engine removes all items directly
- ... or any other SyncML error code, see Reference Manual

## _ENTRY_ void DispItems (CContext *aContext*, bool *allFields*, cAppCharP *specificItem*)

Writes the context of all items to dbg output path This routine is implemented for debug purposes only and will NOT BE CALLED by the SyncML engine. Can be implemented empty, if not needed.

**Parameters:**

<*aContext*> The datastore context.

<*allFields*>

- true : all fields, also empty ones, will be displayed;
- false: only fields <> "" will be shown

<*specificItem*>

- "" : all items will be shown;
- else : shows the <specificItem>

**Returns:**

-

## _ENTRY_ void DisposeObj (CContext *aContext*, void * *memory*)

Disposes memory, which has been allocated within the datastore context. 'DisposeObj' can occur at any time within <aContext>.

**Parameters:**

<*aContext*> The datastore context.

<*memory*> Dispose allocated memory.

**Returns:**

-

### _ENTRY_ TSyError EndDataRead (CContext *aContext*)

This routine terminates the read from database phase It can be used e.g. for termination of a transaction. In standard case it can be implemented empty, returning simply a value LOCERR_OK = 0.

**Parameters:**

<*aContext*> The datastore context.

**Returns:**

error code

### _ENTRY_ TSyError EndDataWrite (CContext *aContext*, bool *success*, appCharP * *newToken*)

Advises the database to finsish the running transaction

**Parameters:**

<*aContext*> The datastore context.

<*success*>

- true: All former actions were successful, so the database can commit
- false: The transaction was not successful, so the database may rollback or ignore the transaction.

<*newToken*> An internally generated string value, which will be used to identify changed database records. It is normally an ISO8601 formatted string, which represents the module's current time (at the time the 'StartDataRead' of this context has been called). All changed records of the currrent context must get this token as timestamp as as well. The SyncML engine will return this value with the 'StartDataRead' call within the next session. It must return NULL in case of no <success>.

**Returns:**

error code, if operation can't be performed. No <success> is not an error.

NOTE: By default, the SyncML engine expects an ISO8601 string for <newToken>. But the SyncML engine can be configured to treat this value completely opaque, if implemented in a different way.

The <newToken> must be allocated locally and will be disposed with a 'DisposeObj' call later by the SyncML engine.

### _ENTRY_ uInt32 FilterSupport (CContext *aContext*, cAppCharP *aFilterRules*)

This function asks for filter support.

**Parameters:**

<*aContext*> The datastore context

<*aFilterRules*> The SyncML sends a list of filter rules. This function has to reply, up to which rule, filters are supported (and switched on now). Data is formatted as multiline aa:bb<CRLF>cc:dd[<CRLF>]

**Returns:**

Up to <n> filters are supported (and switched on) for this context If 0 will be returned, no field of <aFilterRules> are supported.

### _ENTRY_ TSyError FinalizeLocalID (CContext *aContext*, cItemID *aID*, ItemID *updID*)

This routine updates a temporary <aid> to an <updid> at the end For cached systems which assign IDs at the end of a run.

**Parameters:**

<*aContext*> The datastore context.

<*aID*> Database key of dataset to be updated

<*updID*>

- Input: NULL is assigned as default value to <updID.item> and <updID.parent>.
- Output: The updated database key for <aID>. Can be NULL, if the same as <aID>

**Returns:**

error code

- LOCERR_OK ( =0 ), if successful
- DB_Forbidden ( =403 ), if <aItemData> can't be resolved

- DB_NotFound ( =404 ), if unknown <aID>
- LOCERR_NOTIMP ( =20030 ), if no finalizing is needed at all
- ... or any other SyncML error code, see Reference Manual

NOTE: <updID> must either contain NULL references ( if the same as <aID> ), or the memory for <updID.item> must be allocated locally. The SyncML engine will call 'DisposeObj' later for <updID.item> to release the memory. <updID.parent>should be always NULL.

### _ENTRY_ TSyError InsertItem (CContext *aContext*, cAppCharP *aItemData*, ItemID *aID*)

This routine inserts a new dataset to the database. The assigned new **ItemID** <aId> will be returned.

**Parameters:**

<*aContext*> The datastore context.
<*aItemData*> The data, formatted as multiline aa:bb<CRLF>cc:dd[<CRLF>]
<*aID*> Database key of the new dataset.

**Returns:**

error code

- LOCERR_OK ( =0 ), if successful
- DB_DataMerged ( =207 ), if successful, but "ReadItem" requested to inform about updates
- DB_Forbidden ( =403 ), if <aItemData> can't be resolved
- DB_Full ( =420 ), if not enough space in the DB
- ... or any other SyncML error code, see Reference Manual

NOTE: The memory for <aItemID> must be allocated locally. The SyncML engine will call 'DisposeObj' later for <aItemID>, to release the memory

### _ENTRY_ TSyError InsertItemAsKey (CContext *aContext*, KeyH *aItemKey*, ItemID *aID*)

asKey version of “InsertItem”, using <aItemKey>

### _ENTRY_ TSyError InsertMapItem (CContext *aContext*, cMapID *mID*)

Map table handling: Insert a map item of this context

**Parameters:**

<aContext> The datastore context
<mID> **MapID** (with <localID>,<remoteID>, <flags> and <ident> ).
*If there is already a MapID element with <localID> and <ident>, it will be updated, else created.*

**Returns:**

error code, if this **MapID** can't be inserted, or if already existing

USED ONLY WITH <plugin_datastoredadmin>

### _ENTRY_ TSyError LoadAdminData (CContext *aContext*, cAppCharP *aLocalDB*, cAppCharP *aRemoteDB*, appCharP * *adminData*)

This function gets the stored information about the record with the four paramters: <sDevKey>, <sUsrKey>, <aLocalDB>, <aRemoteDB>.

- <plugin_deviceadmin>yes</plugin_deviceadmin>: Admin/Map routines will be used.

**Parameters:**

<*aContext*> The datastore context
<*aLocalDB*> Name of the local DB
<*aRemoteDB*> Name of the remote DB
<*adminData*> The data, saved with the last 'SaveAdminData' call

**Returns:**

error code 404 (NotFound), if record is not (yet) available, 0 (no error) if admin data found

NOTE: <sDevKey> and <sUsrKey> have been passed with 'CreateContext' already. The plug-in module must have stored them within the datastore context.

USED ONLY WITH <plugin_datastoredadmin>

**_ENTRY_ TSyError Module_Capabilities (CContext *mContext*, appCharP * *mCapabilities*)**

Get the module's capabilities Currently the SyncML engine currently understands and supports:

- "plugin_sessionauth"
- "plugin_deviceadmin"
- "plugin_datastoreadmin"
- "plugin_datastore"

If one of these identifiers will be defined as "no" ( e.g. "plugin_sessionauth:no" ), the according routines will not be connected and used.

NOTE: The <mCapabilities> can be allocated with "StrAlloc" (SDK_util.h) for C/C++

**Parameters:**

<*mContext*> The module context.
<*mCapabilities*> Returns the module's capabilities as multiline aa:bb<CRLF>cc:dd[<CRLF>]

**Returns:**

error code

**_ENTRY_ TSyError Module_CreateContext (CContext * *mContext*, cAppCharP *moduleName*, cAppCharP *subName*, cAppCharP *mContextName*, DB_Callback *mCB*)**

Create a module context <mContext> This routine will be called as the 1st or 2nd call, when this module will be connected. (The 1st call is usually a 'Module_Version( 0 )' call outside any context).

It will be called not only once, but for each session and datastore context, as defined at the XML config file. This routine can return error 20028 (LOCERR_ALREADY), if already created. This will be treated not as an error. For this case, it must return the same <mContext> as for the former call(s).

NOTE: The module context can exist once and can be shared for all plug-in accesses. Please note, that write access to such a common module context structure must be thread-safe, when accessed from the session or datastore context. All the 'Module_CreateContext' calls for this module will be called sequentially by one thread. The plug-in programmer is responsible not to re-initialize the context for subsequent calls.

If the module name at the XML config file is defined as "aaa!bbb!ccc" it will be passed as "aaa" to <moduleName> and "bbb!ccc" to <subName>. This mechanism can be used to cascade plug-in modules, where the next module gets "bbb" as <moduleName> and "ccc" as <subName>. The JNI plug-in for Java is using this structure to address the JNI plug-in and its assigned Java class.

**Parameters:**

<*mContext*> Returns a value, which allows to identify this module context. Allowed values: Anything except 0, which is reserved for no context.
<*moduleName*> Name of this plug-in
<*subName*> Name of sub module (if available)
<*mContextName*> Name of the (datastore) context, e.g. "contacts"; this string is empty for calll concerning the session.
<*mCB*> **DB_Callback** structure for module logging

**Returns:**

error code, if context could not be created (e.g. not enough memory), 0 if context successfully created.

### _ENTRY_ TSyError Module_DeleteContext (CContext *mContext*)

This routine will be called as the last call, before this module is disconnected. The SyncML engine will call 'Module_DisposeObj' (if required) before this call

NOTE: This routine will be called ONLY, if the server stops in a controlled way. Its good programming practice not to wait for this 'DeleteContext' call.

**Parameters:**

<*mContext*> The module context.

**Returns:**

error code

### _ENTRY_ void Module_DisposeObj (CContext *mContext*, void * *memory*)

Disposes memory, which has been allocated within the module context. (At the moment this is only the capabilities string). 'Module_DisposeObj' can occur at any time within <mContext>.

NOTE: - If <mCapabilities> has been allocated with "StrAlloc", use "Str_Dispose" (SDK_util.h) to release the memory again.

- If it is defined as const within the plugin module (the module itself knows about !), this routine can be implemented empty.

**Parameters:**

<*mContext*> The module context.
<*memory*> Dispose allocated memory.

**Returns:**

-

### _ENTRY_ TSyError Module_PluginParams (CContext *mContext*, cAppCharP *mConfigParams*, CVersion *engineVersion*)

The module's config params will be sent to the plug-in. It can be used for access path definitions or other things. The <plugin_params> can be defined individually for each session and datastore. The SyncML engine checks the syntax, but not the content. This routine should return an error 20010 (LOCERR_CFGPARSE), if one of these parameters is not supported.

EXAMPLE: Definition at XML config file:

<plugin_params>

<datapath>/var/log/sysync</datapath>

<ultimate_answer>42</ultimate_answer>

</plugin_params>

will be passed as:

"datapath:/var/log/sysync

ultimate_answer:42"

NOTE: Module_PluginParams will be called ALWAYS for each module context, even if no plug-in parameter is defined. This allows to react consistently on parameters, which are not always available.

**Parameters:**

<*mContext*> The module context.
<*mConfigParams*> The plugin params as multiline aa:bb<CRLF>cc:dd[<CRLF>]
<*engineVersion*> The SyncML engine's version

**Returns:**

error code

### _ENTRY_ CVersion Module_Version (CContext *mContext*)

Get the module's version.

NOTE: The SyncML will take decisions depending on this version number, so the plug-in developer should not change the values at the delivered sample code. **Plugin_Version( short buildNumber )** of 'SDK_util' should be used. The <buildNumber> can be defined by the user.

NOTE: This function can be called by the engine outside any context with <mContext> = 0. For this case, any callback is not permitted (as no **DB_Callback** is available).

**Parameters:**

<*mContext*> The module context ( 0, if none ).

**Returns:**

current version as SDK_VERSION_MAJOR | SDK_VERSION_MINOR) | SDK_SUBVERSION | buildNumber

### _ENTRY_ TSyError MoveItem (CContext *aContext*, cItemID *aID*, cAppCharP *newParentID*)

This routine moves <aID.item> from <aID.parent> to <newParentID>

**Parameters:**

<*aContext*> The datastore context.
<*aID*> **ItemID** ( with <item>,<parent> ) to be moved.
<*newParentID*> New parent ID for <aID>

**Returns:**

error code

- LOCERR_OK ( =0 ), if successful
- DB_NotFound ( =404 ), if unknown <newParentID>
- DB_Full ( =420 ), if not enough space in the DB
- ... or any other SyncML error code, see Reference Manual

### _ENTRY_ TSyError ReadBlob (CContext *aContext*, const ItemID *aID*, cAppCharP *aBlobID*, appPointer **aBlkPtr*, memSize **aBlkSize*, memSize **aTotSize*, bool *aFirst*, bool **aLast*)

This routine reads the specific binary logic block <aID>,<aBlobID> from the database.

**Parameters:**

<*aContext*> The datastore context.
<*aID*> **ItemID** ( with <item>,<parent> ).
<*aBlobID*> The assigned ID of the blob.
<*aBlkPtr*> Position and size (in bytes) of the blob block.
<*aBlkSize*>

- Input: Maximum size (in bytes) of the blob block to be read. If <blkSize> is 0, the result size is not limited.
- Output: Size (in bytes) of the blob block. <blkSize> must not be larger than its input value.

<*aTotSize*> Total size of the blob (in bytes), can be also 0, if not available, e.g. for a stream.
<*aFirst*> (Input)

- true : Engine asks for the first block of this blob.
- false: Engine asks for the next block of this blob.

<*aLast*> (Output)

- true : This is the last part (or the whole) blob.
- false: More blocks will follow.

**Returns:**

error code, if not ok ( e.g. invalid <aItemID>,<aBlobID> )

NOTE 1) The memory at <blkPtr>,<blkSize> must be allocated locally. The SyncML engine will call 'DisposeObj' later for <blkPtr>, to release the memory.

NOTE 2) Empty blobs are allowed, <blkSize> and <totSize> must be set to 0, <blkPtr> can be undefined, <aLast> must be true. No 'DisposeObj' call is required for this case.

NOTE 3) The SyncML engine can change to read another blob before having read the whole blob. It will never resume reading of this incomplete blob, but start reading again with <aFirst> = true.

**_ENTRY_ TSyError ReadItem (CContext *aContext*, cItemID *aID*, appCharP * *aItemData*)**

This routine reads the contents of a specific **ItemID** <aID> from the database.

**Parameters:**

<*aContext*> The datastore context.
<*aID*> The assigned **ItemID** in the database
<*aItemData*> Returns the data, formatted as multiline aa:bb<CRLF>cc:dd[<CRLF>]

**Returns:**

error code, if not ok ( e.g. invalid <aItemID> )

NOTE: The memory for <aItemData> must be allocated locally. The SyncML engine will call 'DisposeObj' later for <aItemData>, to release again its memory.

**_ENTRY_ TSyError ReadItemAsKey (CContext *aContext*, cItemID *aID*, KeyH *aItemKey*)**

asKey version of "ReadItem", using <aItemKey>

**_ENTRY_ TSyError ReadNextItem (CContext *aContext*, ItemID *aID*, appCharP * *aItemData*, sInt32 * *aStatus*, bool *aFirst*)**

This routine reads the next **ItemID** from the database. <allfields> of 'ContextSupport' ( "ReadNextItem:allfields" ) and <aFilterRules> of 'FilterSupport' must be considered. If <aFirst> is true, the routine must return the first element (again).

**Parameters:**

<*aContext*> The datastore context.
<*aID*> The assigned **ItemID** in the database; will be ignored by the SyncML engine, if <aStatus> = 0
<*aItemData*> The data, formatted as multiline aa:bb<CRLF>cc:dd[<CRLF>]; will be ignored by the SyncML engine, if <aStatus> = 0
<*aStatus*>

- ReadItem_EOF ( =0 ) for none ( =eof ),
- ReadItem_Changed ( =1 ) for a changed item,
- ReadItem_Unchanged ( =2 ) for unchanged item.
- ReadItem_Resumed ( =3 ) for a changed item (since resumed)

<*aFirst*>

- true: the routine must return the first element
- false: the routine must return the next element

**Returns:**

error code, if not ok. No datasets found is a success as well !

NOTE: The memory for <aID> and <aItemData> must be allocated locally. The SyncML engine will call 'DisposeObj' later for these objects to release the memory again. It needn't to be allocated, if <aStatus> is ReadItem_EOF.

NOTE: By default, the SyncML engine asks for <aID> only. <aItemData> can be returned, if anyway available or <aItemData> must be returned, if the engine asks for it (when calling "ReadNextItem:allfields" at 'ContextSupport' with <allfields>).

**_ENTRY_ TSyError ReadNextItemAsKey (CContext *aContext*, ItemID *aID*, KeyH *aItemKey*, sInt32 * *aStatus*, bool *aFirst*)**

asKey version of "ReadNextItem", using <aItemKey>

**_ENTRY_ bool ReadNextMapItem (CContext *aContext*, MapID *mID*, bool *aFirst*)**

Map table handling: Get the next map item of this context. If <aFirst> is true, the routine must start to return the first element

**Parameters:**

<*aContext*> The datastore context
<*mID*> **MapID** ( with <localID>,<remoteID> and <flags> ).
<*aFirst*> Starting with the first **MapID**. When creating a context, the first call will get the first **MapID**, even if <aFirst> is false.

**Returns:**

- true: as long as there is a **MapID** available, which must be assigned to <mID>
- false: if there is no more **MapID**. Nothing must be assigned to <mID>

USED ONLY WITH <plugin_datastoredadmin>

**_ENTRY_ TSyError SaveAdminData (CContext *aContext*, cAppCharP *adminData*)**

This functions stores the new <adminData> for this context

**Parameters:**

<*aContext*> The datastore context
<*adminData*> The new set of admin data to be stored, will be loaded again with the next 'LoadAdmin-Data' call.

**Returns:**

error code, if data could not be saved (e.g. not enough memory); 0 if successfully created.

USED ONLY WITH <plugin_datastoredadmin>

**_ENTRY_ TSyError Session_AdaptItem (CContext *sContext*, appCharP * *sItemData1*, appCharP * *sItemData2*, appCharP * *sLocalVars*, uInt32 *sIdentifier*)**

This function adapts itemData

**Parameters:**

<*sContext*> The session context
<*sItemData1*> The 1st item's data
<*sItemData2*> The 2nd item's data
<*sLocalVars*> The local vars
<*sIdentifier*> To identify, where it is called

**Returns:**

error code

NOTE: The memory for adapted strings must be allocated locally. The SyncML engine will call 'DisposeObj' later, to release again its memory. One or more strings can be returned unchanged as well.

**_ENTRY_ TSyError Session_CheckDevice (CContext *sContext*, cAppCharP *aDeviceID*, appCharP * *sDevKey*, appCharP * *nonce*)**

Check the database entry of <aDeviceID> and return its <nonce> string. If <aDeviceID> is not yet available at the plug-in, return "" for <nonce>

**Parameters:**

<*sContext*> The session context
<*aDeviceID*> The assigned device ID string
<*sDevKey*> The device key string (will be used for datastore accesses later)
<*nonce*> The nonce string of the last session If <aDeviceID> is not yet available, return "" for <nonce> and error code 0.

**Returns:**

error code 403 (Forbidden), if plugin_deviceadmin is not supported; 0, if successful

USED ONLY WITH <plugin_deviceadmin>

**_ENTRY_ TSyError Session_CreateContext (CContext * *sContext*, cAppCharP *sessionName*, DB_Callback *sCB*)**

By default the session context will be handled by the ODBC interface. The session context of this plug-in module will be used only, if <server type="plugin"> and <plugin_module> is defined ( <plugin_module>name_of_the_plugin</plugin_module> ). <plugin_params> can be defined individually.

**Parameters:**

<*sContext*> Returns a value, which allows to identify this session context.
<*sessionName*> Name of this session
<*sCB*> **DB_Callback** structure for session logging

**Returns:**

error code, if context could not be created (e.g. not enough memory) 0 if context successfully created,

Flags (at the XML config file):

- <plugin_deviceadmin>yes</plugin_deviceadmin>: "Session_CheckDevice", "Session_GetNonce" "Session_SaveNonce" and "Session_SaveDeviceInfo" will be used.

- <plugin_sessionauth>yes</plugin_sessionauth>: "Session_PasswordMode", "Session_Login" and "Session_Logout" will be used.

**_ENTRY_ TSyError Session_DeleteContext (CContext *sContext*)**

Delete a session context. No access to <sContext> will be done after this call

**Parameters:**

<*sContext*> The session context

**Returns:**

error code, if context could not be deleted.

**_ENTRY_ void Session_DispItems (CContext *sContext*, bool *allFields*, cAppCharP *specificItem*)**

Writes the context of all items to dbg output path This routine is implemented for debug purposes only and will NOT BE CALLED by the SyncML engine. Can be implemented empty

**Parameters:**

<*sContext*> The session context
<*allFields*> true : all fields, also empty ones, will be displayed; false: only fields <> "" will be shown
<*specificItem*> "" : all items will be shown; else shows the <specificItem>

**Returns:**

-

**_ENTRY_ void Session_DisposeObj (CContext *sContext*, void * *memory*)**

Disposes memory, which has been allocated within the session context. 'Session_DisposeObj' can occur at any time within <sContext>.

**Parameters:**

<*sContext*> The session context.
<*memory*> Dispose allocated memory.

**Returns:**

-

**_ENTRY_ TSyError Session_GetDBTime (CContext *sContext*, appCharP * *currentDBTime*)**

Get the current DB time of <sContext>

**Parameters:**

<*sContext*> The session context
<*currentDBTime*> The current time of the plugin's DB (as ISO8601 format).

**Returns:**

error code 403 (Forbidden), if plugin_deviceadmin is not supported; 404 (NotFound), if not available -> the engine creates its own time 0, if successful

### _ENTRY_ TSyError Session_GetNonce (CContext *sContext*, appCharP * *nonce*)

Get a new nonce from the database. If this routine returns an error, the SyncML engine will create its own nonce.

**Parameters:**

<*sContext*> The session context
<*nonce*> A valid new nonce value (for the assigned device ID).

**Returns:**

error code 404 (NotFound), if no <nonce> has been generated; 0, if a valid <nonce> has been generated

USED ONLY WITH <plugin_deviceadmin>

### _ENTRY_ TSyError Session_Login (CContext *sContext*, cAppCharP *sUsername*, appCharP * *sPassword*, appCharP * *sUsrKey*)

Get <sUsrKey> of <sUsername>,<sPassword> in the session context.

**Parameters:**

<*sContext*> The session context
<*sUsername*> The user name ...
<*sPassword*> ... and the password. <sPassword> is an input parameter for 'Password_ClrTxt_IN' mode and an output parameter for 'Password_ClrText_OUT' and 'Password_MD5_OUT' modes.
<*sUsrKey*> Returns the internal reference key, which will be passed to to the datastore contexts later.

**Returns:**

error code 403 (Forbidden), if plugin_sessionauth is not supported; 0, if successful

USED ONLY WITH <plugin_sessionauth>

### _ENTRY_ TSyError Session_Logout (CContext *sContext*)

Logout for this session context

**Parameters:**

<*sContext*> The session context

**Returns:**

error code 403 (Forbidden), if plugin_sessionauth is not supported; 0, if successful

USED ONLY WITH <plugin_sessionauth>

### _ENTRY_ sInt32 Session_PasswordMode (CContext *sContext*)

Get the password mode. There are currently 4 different password modes supported.

**Parameters:**

<*sContext*> The session context

**Returns:**

- Password_ClrText_IN : 'SessionLogin' will get clear text password
- Password_ClrText_OUT : '' must return clear text password
- Password_MD5_OUT : '' must return MD5 coded password
- Password_MD5_Nonce_IN: '' will get MD5B64(MD5B64(user:pwd):nonce)

USED ONLY WITH <plugin_sessionauth>

**_ENTRY_ TSyError Session_SaveDeviceInfo (CContext *sContext*, cAppCharP *aDeviceInfo*)**

Save the device info for <sContext>

**Parameters:**

<*sContext*> The session context

<*aDeviceInfo*> More information about the assigned device (for DB and logging)

**Returns:**

error code 403 (Forbidden), if plugin_deviceadmin is not supported; 0, if successful

USED ONLY WITH <plugin_deviceadmin>

**_ENTRY_ TSyError Session_SaveNonce (CContext *sContext*, cAppCharP *nonce*)**

Save the new nonce (which will be expected to be returned in the next session for this device ID.

**Parameters:**

<*sContext*> The session context

<*nonce*> New <nonce> for the next session (of the assigned device ID)

**Returns:**

error code 403 (Forbidden), if plugin_deviceadmin is not supported; 0, if successful

USED ONLY WITH <plugin_deviceadmin>

**_ENTRY_ void Session_ThreadMayChangeNow (CContext *sContext*)**

Due to the architecture of the SyncML engine, the system may run in a multithread environment. The consequence is that each routine of this plugin module can be called by a different thread. Normally this is not a problem, nevertheless this routine notifies about thread changes in <sContext>. It can be ignored ( =implemented empty), if not really needed.

**Parameters:**

<*sContext*> The session context

**Returns:**

-

**_ENTRY_ TSyError StartDataRead (CContext *aContext*, cAppCharP *lastToken*, cAppCharP *resumeToken*)**

This routine initializes reading from the database StartDataRead must prepare the database to return the objects of this context.

**Parameters:**

<*aContext*> The datastore context.

<*lastToken*> The value which has been returned by this module at the last "EndDataWrite" call will be given. It will be "", when called the first time. Normally this token is an ISO8601 formatted string which represents the module's current time (at the beginning of a session). It will be used to decide at 'ReadNextItem' whether a record has been changed.

<*resumeToken*> Token for Suspend/Resume mode.

**Returns:**

error code

**_ENTRY_ TSyError StartDataWrite (CContext *aContext*)**

This routine initializes writing to the database

**Parameters:**

<*aContext*> The datastore context.

**Returns:**

error code, if not ok (e.g. invalid select options)

**_ENTRY_ void ThreadMayChangeNow (CContext *aContext*)**

Due to the architecture of the SyncML engine, the system may run in a multithread environment. The consequence is that each routine of this API module can be called by a different thread. Normally this is not a problem, nevertheless this routine notifies about thread changes in <aContext>. It can be ignored ( =implemented empty), if not really needed.

**Parameters:**

<*aContext*> The datastore context.

**Returns:**

-

**_ENTRY_ TSyError UpdateItem (CContext *aContext*, cAppCharP *aItemData*, cItemID *aID*, ItemID *updID*)**

This routine updates an existing dataset of the database

**Parameters:**

<*aContext*> The datastore context.
<*aItemData*> The data, formatted as multiline aa:bb<CRLF>cc:dd[<CRLF>]
<*aID*> Database key of dataset to be updated
<*updID*>

- Input: NULL is assigned as default value to <updID.item> and <updID.parent>.
- Output: The updated database key for <aID>. Can be NULL, if the same as <aID>

**Returns:**

error code

- LOCERR_OK ( =0 ), if successful
- DB_Forbidden ( =403 ), if <aItemData> can't be resolved
- DB_NotFound ( =404 ), if unknown <aID>
- DB_Full ( =420 ), if not enough space in the DB
- ... or any other SyncML error code, see Reference Manual

NOTE: <updID> must either contain NULL references ( if the same as <aID> ), or the memory for <updID.item>,<updID.parent> must be allocated locally. The SyncML engine will call 'DisposeObj' later for <updID.item> and <updID.parent> to release the memory. <updID.parent> can be NULL, if the hierarchical model is not supported.

**_ENTRY_ TSyError UpdateItemAsKey (CContext *aContext*, KeyH *aItemKey*, cItemID *aID*, ItemID *updID*)**

asKey version of "UpdateItem", using <aItemKey>

**_ENTRY_ TSyError UpdateMapItem (CContext *aContext*, cMapID *mID*)**

Map table handling: Update a map item of this context

**Parameters:**

<*aContext*> The datastore context
<*mID* **MapID** ( with <localID>,<remoteID>, <flags> and <ident> ).

**Returns:**

error code, if this **MapID** can't be updated or if it does not exist.

USED ONLY WITH <plugin_datastoredadmin>

**_ENTRY_ TSyError WriteBlob (CContext *aContext*, const ItemID *aID*, cAppCharP *aBlobID*, appPointer *aBlkPtr*, memSize *aBlkSize*, memSize *aTotSize*, bool *aFirst*, bool *aLast*)**

This routine writes the specific binary logic block <blobID> to the database.

**Parameters:**

<*aContext*> The datastore context.
<*aID*> **ItemID** ( with <item>,<parent> ).
<*aBlobID*> The assigned ID of the blob.
<*aBlkPtr*>
<*aBlkSize*> Position and size (in bytes) of the blob block.
<*aTotSize*> Total size of the blob (in bytes), Can be also 0, if not available, e.g. for a stream.
<*aFirst*>

- true : this is the first block of the blob.
- false: this is the next block.

<*aLast*>

- true : this is the last block.
- false: more blocks will follow.

**Returns:**

error code, if not ok ( e.g. invalid <aID>,<aBlobID> )

NOTE: Empty blobs are possible, <blkSize> and <totSize> will be set to 0, <blkPtr> will be NULL, <aFirst> and <aLast> will be true.

**_ENTRY_ void WriteLogData (CContext *aContext*, cAppCharP *logData*)**

This functions writes <logData> for this context Can be implemented empty, if not needed.

**Parameters:**

<*aContext*> The datastore context.
<*logData*> Logging information, formatted as multiline aa:bb<CRLF>cc:dd[<CRLF>]

**Returns:**

-

# 9. UIApi Interface description

## 9.1 Functions in the UI_Call_In call-in structure

The following list only shows the function prototypes. These are required when accessing the UIApi from plain C. See documentation of the **TEngineModuleBase** class members for details. The routines shown here are all also implemented as methods of **TEngineModuleBase** (and are available in the wrapper class **TEngineModuleBridge** which facilitates access from C++ code and is part of the SDK), and have similar signatures (but no aCB first argument because the callback structure is a class member and using by-reference arguments instead of plain pointers where appropriate).

- void **DebugDB** (void *aCB, cAppCharP aParams)
- void **DebugExotic** (void *aCB, cAppCharP aParams)
- void **DebugBlock** (void *aCB, cAppCharP aTag, cAppCharP aDesc, cAppCharP aAttrText)
- void **DebugEndBlock** (void *aCB, cAppCharP aTag)
- void **DebugEndThread** (void *aCB)

- TSyError **SetStringMode** (void *aCB, uInt16 aCharSet, uInt16 aLineEndMode, bool aBigEndian)
- TSyError **InitEngineXML** (void *aCB, cAppCharP aConfigXML)
- TSyError **InitEngineFile** (void *aCB, cAppCharP aConfigFilePath)
- TSyError **InitEngineCB** (void *aCB, TXMLConfigReadFunc aReaderFunc, void *aContext)
- TSyError **OpenSession** (void *aCB, **SessionH** *aSessionH, uInt32 aSelector, cAppCharP aSessionName)
- TSyError **OpenSessionKey** (void *aCB, **SessionH** aSessionH, **KeyH** *aKeyH, uInt16 aMode)
- TSyError **SessionStep** (void *aCB, **SessionH** aSessionH, uInt16 *aStepCmd, TEngineProgressInfo *aInfoP)
- TSyError **GetSyncMLBuffer** (void *aCB, **SessionH** aSessionH, bool aForSend, appPointer *aBuffer, memSize *aBufSize)
- TSyError **RetSyncMLBuffer** (void *aCB, **SessionH** aSessionH, bool aForSend, memSize aRetSize)
- TSyError **ReadSyncMLBuffer** (void *aCB, **SessionH** aSessionH, appPointer aBuffer, memSize aBufSize, memSize *aValSize)
- TSyError **WriteSyncMLBuffer** (void *aCB, **SessionH** aSessionH, appPointer aBuffer, memSize aValSize)
- TSyError **CloseSession** (void *aCB, **SessionH** aSessionH)
- TSyError **OpenKeyByPath** (void *aCB, **KeyH** *aKeyH, **KeyH** aParentKeyH, cAppCharP aPath, uInt16 aMode)
- TSyError **OpenSubkey** (void *aCB, **KeyH** *aKeyH, **KeyH** aParentKeyH, sInt32 aID, uInt16 aMode)
- TSyError **DeleteSubkey** (void *aCB, **KeyH** aParentKeyH, sInt32 aID)
- TSyError **GetKeyID** (void *aCB, **KeyH** aKeyH, sInt32 *aID)
- TSyError **SetTextMode** (void *aCB, **KeyH** aKeyH, uInt16 aCharSet, uInt16 aLineEndMode, bool aBigEndian)
- TSyError **SetTimeMode** (void *aCB, **KeyH** aKeyH, uInt16 aTimeMode)
- TSyError **CloseKey** (void *aCB, **KeyH** aKeyH)
- TSyError **GetValue** (void *aCB, **KeyH** aKeyH, cAppCharP aValName, uInt16 aValType, appPointer aBuffer, memSize aBufSize, memSize *aValSize)
- TSyError **GetValueByID** (void *aCB, **KeyH** aKeyH, sInt32 aID, sInt32 aArrayIndex, uInt16 aValType, appPointer aBuffer, memSize aBufSize, memSize *aValSize)
- sInt32 **GetValueID** (void *aCB, **KeyH** aKeyH, cAppCharP aName)

- TSyError **SetValue** (void *aCB, **KeyH** aKeyH, cAppCharP aValName, uInt16 aValType, cAppPointer aBuffer, memSize aValSize)
- TSyError **SetValueByID** (void *aCB, **KeyH** aKeyH, sInt32 aID, sInt32 aArrayIndex, uInt16 aValType, cAppPointer aBuffer, memSize aValSize)

## 9.2 TEngineModuleBase Class Reference

### *9.2.1 Public Member Function Overview*

- **TEngineModuleBase** ()
- virtual **~TEngineModuleBase** ()
- TSyError **Connect** (string aEngineName, unsigned long aPrgVersion=0, unsigned short aDebug-Flags=DBG_PLUGIN_NONE)
- virtual TSyError **Init** ()=0
- virtual TSyError **SetStringMode** (uInt16 aCharSet, uInt16 aLineEndMode=LEM_CSTR, bool aBigEndian=false)=0

  *Set the global mode for string paramaters (when never called, default params are UTF-8 with C-style line ends).*
- virtual TSyError **InitEngineXML** (cAppCharP aConfigXML)=0

  *init object, optionally passing XML config text in memory*
- virtual TSyError **InitEngineFile** (cAppCharP aConfigFilePath)=0

  *init object, optionally passing a open FILE for reading config*
- virtual TSyError **InitEngineCB** (**TXMLConfigReadFunc** aReaderFunc, void *aContext)=0

  *init object, optionally passing a callback for reading config*
- virtual TSyError **OpenSession** (**SessionH** &aNewSessionH, uInt32 aSelector=0, cAppCharP aSession-Name=NULL)=0

  *Open a session.*
- virtual TSyError **OpenSessionKey** (**SessionH** aSessionH, **KeyH** &aNewKeyH, uInt16 aMode)=0

  *open session specific runtime parameter/settings key*
- virtual TSyError **SessionStep** (**SessionH** aSessionH, uInt16 &aStepCmd, **TEngineProgressInfo** *aInfoP=NULL)=0

  *Executes sync session or other sync related activity step by step.*
- virtual TSyError **GetSyncMLBuffer** (**SessionH** aSessionH, bool aForSend, appPointer &aBuffer, memSize &aBufSize)=0

  *Get access to SyncML message buffer.*
- virtual TSyError **RetSyncMLBuffer** (**SessionH** aSessionH, bool aForSend, memSize aProcessed)=0

  *Return SyncML message buffer to engine.*
- virtual TSyError **ReadSyncMLBuffer** (**SessionH** aSessionH, appPointer aBuffer, memSize aBufSize, memSize &aMsgSize)=0

  *Read data from SyncML message buffer.*
- virtual TSyError **WriteSyncMLBuffer** (**SessionH** aSessionH, appPointer aBuffer, memSize aMsgSize)=0

  *Write data to SyncML message buffer.*
- virtual TSyError **CloseSession** (**SessionH** aSessionH)=0

  *Close a session.*
- virtual TSyError **OpenKeyByPath** (**KeyH** &aNewKeyH, **KeyH** aParentKeyH, cAppCharP aPath, uInt16 aMode)=0

*open Settings key by path specification*

- virtual TSyError **OpenSubkey** (**KeyH** &aNewKeyH, **KeyH** aParentKeyH, sInt32 aID, uInt16 aMode)=0

  *open Settings subkey key by ID or iterating over all subkeys*

- virtual TSyError **DeleteSubkey** (**KeyH** aParentKeyH, sInt32 aID)=0

  *delete Settings subkey key by ID*

- virtual TSyError **GetKeyID** (**KeyH** aKeyH, sInt32 &aID)=0

  *Get key ID of currently open key. Note that the Key ID is only locally unique within the parent key.*

- virtual TSyError **SetTextMode** (**KeyH** aKeyH, uInt16 aCharSet, uInt16 aLineEndMode=LEM_CSTR, bool aBigEndian=false)=0

  *Set text format parameters (when never called, default params are those set with global SetStringMode()).*

- virtual TSyError **SetTimeMode** (**KeyH** aKeyH, uInt16 aTimeMode)=0

  *Set time format parameters.*

- virtual TSyError **CloseKey** (**KeyH** aKeyH)=0

  *Closes a key opened by OpenKeyByPath() or OpenSubKey().*

- virtual TSyError **GetValue** (**KeyH** aKeyH, cAppCharP aValName, uInt16 aValType, appPointer aBuffer, memSize aBufSize, memSize &aValSize)=0

  *Reads a named value in specified format into passed memory buffer.*

- virtual sInt32 **GetValueID** (**KeyH** aKeyH, cAppCharP aName)=0

  *get value's ID for use with Get/SetValueByID()*

- virtual TSyError **GetValueByID** (**KeyH** aKeyH, sInt32 aID, sInt32 aArrayIndex, uInt16 aValType, appPointer aBuffer, memSize aBufSize, memSize &aValSize)=0

  *Reads a named value in specified format into passed memory buffer.*

- virtual TSyError **SetValue** (**KeyH** aKeyH, cAppCharP aValName, uInt16 aValType, cAppPointer aBuffer, memSize aValSize)=0

  *Writes a named value in specified format passed in memory buffer.*

- virtual TSyError **SetValueByID** (**KeyH** aKeyH, sInt32 aID, sInt32 aArrayIndex, uInt16 aValType, cAppPointer aBuffer, memSize aValSize)=0

  *Writes a named value in specified format passed in memory buffer.*

- TSyError **CloseKeyAndNULL** (**KeyH** &aKeyH)

  *Closes a key and nulls the handle.*

- TSyError **CloseSessionAndNULL** (**SessionH** &aSessionH)

  *Closes a session and nulls the handle.*

## 9.2.2 Member Function Documentation

**TSyError sysync::TEngineModuleBase::Connect (string *aEngineName*, unsigned long *aPrgVersion* = `0`, unsigned short *aDebugFlags* = `DBG_PLUGIN_NONE`)**

**virtual TSyError sysync::TEngineModuleBase::SetStringMode (uInt16 *aCharSet*, uInt16 *aLineEndMode* = `LEM_CSTR`, bool *aBigEndian* = `false`) `[pure virtual]`**

Set the global mode for string paramaters (when never called, default params are UTF-8 with C-style line ends).

**Parameters:**

*aCharSet[in]* charset
*aLineEndMode[in]* line end mode (default is C-lineends of the platform (almost always LF))
*aBigEndian[in]* determines endianness of UTF16 text (defaults to little endian = intel order)

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::InitEngineXML (cAppCharP *aConfigXML*) `[pure virtual]`**

init object, optionally passing XML config text in memory

**Parameters:**

*aConfigXML[in]* NULL or empty string if no external config needed, config text otherwise

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::InitEngineFile (cAppCharP *aConfigFilePath*) `[pure virtual]`**

init object, optionally passing a open FILE for reading config

**Parameters:**

*aConfigFilePath[in]* path to config file

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::InitEngineCB (TXMLConfigReadFunc *aReaderFunc*, void * *aContext*) `[pure virtual]`**

init object, optionally passing a callback for reading config

**Parameters:**

*aReaderFunc[in]* callback function which can deliver next chunk of XML config data
*aContext[in]* free context pointer passed back with callback

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::OpenSession (SessionH & *aNewSessionH*, uInt32 *aSelector* = 0, cAppCharP *aSessionName* = NULL) [pure virtual]**

Open a session.

**Parameters:**

*aNewSessionH[out]* receives session handle for all session execution calls
*aSelector[in]* selector, depending on session type. For multi-profile clients: profile ID to use
*aSessionName[in]* a text name/id to identify a session, useage depending on session type.

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::OpenSessionKey (SessionH *aSessionH*, KeyH & *aNewKeyH*, uInt16 *aMode*) [pure virtual]**

open session specific runtime parameter/settings key

**Note:**

key handle obtained with this call must be closed BEFORE SESSION IS CLOSED!

**Parameters:**

*aSessionH[in]* session handle obtained with OpenSession
*aNewKeyH[out]* receives the opened key's handle on success
*aMode[in]* the open mode

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::SessionStep (SessionH *aSessionH*, uInt16 & *aStepCmd*, TEngineProgressInfo * *aInfoP* = NULL) [pure virtual]**

Executes sync session or other sync related activity step by step.

**Parameters:**

*aSessionH[in]* session handle obtained with OpenSession
*aStepCmd[in/out]* step command (STEPCMD_xxx):

- tells caller to send or receive data or end the session etc.
- instructs engine to suspend or abort the session etc.

*aInfoP[in]* pointer to a **TEngineProgressInfo** structure, NULL if no progress info needed

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::GetSyncMLBuffer (SessionH *aSessionH*, bool *aForSend*, appPointer & *aBuffer*, memSize & *aBufSize*) [pure virtual]**

Get access to SyncML message buffer.

**Parameters:**

*aSessionH[in]* session handle obtained with OpenSession
*aForSend[in]* direction send/receive
*aBuffer[out]* receives pointer to buffer (empty for receive, full for send)
*aBufSize[out]* receives size of empty or full buffer

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::RetSyncMLBuffer (SessionH *aSessionH*, bool *aForSend*, memSize *aProcessed*) `[pure virtual]`**

Return SyncML message buffer to engine.

**Parameters:**
*aSessionH[in]* session handle obtained with OpenSession
*aForSend[in]* direction send/receive
*aProcessed[in]* number of bytes put into or read from the buffer
**Returns:**
LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::ReadSyncMLBuffer (SessionH *aSessionH*, appPointer *aBuffer*, memSize *aBufSize*, memSize & *aMsgSize*) `[pure virtual]`**

Read data from SyncML message buffer.

**Parameters:**
*aSessionH[in]* session handle obtained with OpenSession
*aBuffer[in]* pointer to buffer
*aBufSize[in]* size of buffer, maximum to be read
*aMsgSize[out]* size of data available in the buffer for read INCLUDING just returned data.
**Note:**
If the aBufSize is too small to return all available data LOCERR_TRUNCATED will be returned, and the caller can repeat calls to ReadSyncMLBuffer to get the next chunk.
**Returns:**
LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::WriteSyncMLBuffer (SessionH *aSessionH*, appPointer *aBuffer*, memSize *aMsgSize*) `[pure virtual]`**

Write data to SyncML message buffer.

**Parameters:**
*aSessionH[in]* session handle obtained with OpenSession
*aBuffer[in]* pointer to buffer
*aMsgSize[in]* size of message to write to the buffer
**Returns:**
LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::CloseSession (SessionH *aSessionH*) `[pure virtual]`**

Close a session.

**Note:**
It depends on session type if this also destroys the session or if it may persist and can be re-opened.
**Parameters:**
*aSessionH[in]* session handle obtained with OpenSession
**Returns:**
LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::OpenKeyByPath (KeyH & *aNewKeyH*, KeyH *aParentKeyH*, cAppCharP *aPath*, uInt16 *aMode*) `[pure virtual]`**

open Settings key by path specification

**Parameters:**

*aNewKeyH[out]* receives the opened key's handle on success
*aParentKeyH[in]* NULL if path is absolute from root, handle to an open key for relative access
*aPath[in]* the path specification as null terminated string
*aMode[in]* the open mode

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::OpenSubkey (KeyH & *aNewKeyH*, KeyH *aParentKeyH*, sInt32 *aID*, uInt16 *aMode*) `[pure virtual]`**

open Settings subkey key by ID or iterating over all subkeys

**Parameters:**

*aNewKeyH[out]* receives the opened key's handle on success
*aParentKeyH[in]* handle to the parent key
*aID[in]* the ID of the subkey to open, or KEYVAL_ID_FIRST/KEYVAL_ID_NEXT to iterate over existing subkeys or KEYVAL_ID_NEW to create a new subkey
*aMode[in]* the open mode

**Returns:**

LOCERR_OK on success, DB_NoContent when no more subkeys are found with KEYVAL_ID_FIRST/KEYVAL_ID_NEXT or any other SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::DeleteSubkey (KeyH *aParentKeyH*, sInt32 *aID*) `[pure virtual]`**

delete Settings subkey key by ID

**Parameters:**

*aParentKeyH[in]* handle to the parent key
*aID[in]* the ID of the subkey to delete

**Returns:**

LOCERR_OK on success or any other SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::GetKeyID (KeyH *aKeyH*, sInt32 & *aID*) `[pure virtual]`**

Get key ID of currently open key. Note that the Key ID is only locally unique within the parent key.

**Parameters:**

*aKeyH[in]* an open key handle
*aID[out]* receives the ID of the open key, which can be used to re-access the key within its parent using **OpenSubkey()**

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::SetTextMode (KeyH *aKeyH*, uInt16 *aCharSet*, uInt16 *aLineEndMode* = `LEM_CSTR`, bool *aBigEndian* = `false`) `[pure virtual]`**

Set text format parameters (when never called, default params are those set with global **SetStringMode()**).

**Parameters:**

*aKeyH[in]* an open key handle
*aCharSet[in]* charset
*aLineEndMode[in]* line end mode (defaults to C-lineends of the platform (almost always LF))
*aBigEndian[in]* determines endianness of UTF16 text (defaults to little endian = intel order)

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::SetTimeMode (KeyH *aKeyH*, uInt16 *aTimeMode*) `[pure virtual]`**

Set time format parameters.

**Parameters:**

*aKeyH[in]* an open key handle
*aTimeMode[in]* time mode, see TMODE_xxx (default is platform's lineratime_t when **SetTimeMode()** is not used)

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::CloseKey (KeyH *aKeyH*) `[pure virtual]`**

Closes a key opened by **OpenKeyByPath()** or OpenSubKey().

**Parameters:**

*aKeyH[in]* an open key handle. Will be invalid when call returns with LOCERR_OK. Do not re-use!

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::GetValue (KeyH *aKeyH*, cAppCharP *aValName*, uInt16 *aValType*, appPointer *aBuffer*, memSize *aBufSize*, memSize & *aValSize*) `[pure virtual]`**

Reads a named value in specified format into passed memory buffer.

**Parameters:**

*aKeyH[in]* an open key handle
*aValName[in]* name of the value to read
*aValType[in]* desired return type, see VALTYPE_xxxx
*aBuffer[in/out]* buffer where to store the data
*aBufSize[in]* size of buffer in bytes (ALWAYS in bytes, even if value is Unicode string)
*aValSize[out]* actual size of value. For VALTYPE_TEXT, size is string length (IN BYTES) excluding NULL terminator Note that this will be set also when return value is LOCERR_BUFTOOSMALL, to indicate the required buffer size

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual sInt32 sysync::TEngineModuleBase::GetValueID (KeyH *aKeyH*, cAppCharP *aName*) [pure virtual]**

get value's ID for use with Get/SetValueByID()

**Returns:**
KEYVAL_ID_UNKNOWN when no ID available for name, ID of value otherwise

**virtual TSyError sysync::TEngineModuleBase::GetValueByID (KeyH *aKeyH*, sInt32 *aID*, sInt32 *aArrayIndex*, uInt16 *aValType*, appPointer *aBuffer*, memSize *aBufSize*, memSize & *aValSize*) [pure virtual]**

Reads a named value in specified format into passed memory buffer.

**Parameters:**
*aKeyH[in]* an open key handle
*aID[in]* ID of the value to read
*aArrayIndex[in]* 0-based array element index for array values.
*aValType[in]* desired return type, see VALTYPE_xxxx
*aBuffer[in/out]* buffer where to store the data
*aBufSize[in]* size of buffer in bytes (ALWAYS in bytes, even if value is Unicode string)
*aValSize[out]* actual size of value. For VALTYPE_TEXT, size is string length (IN BYTES) excluding NULL terminator Note that this will be set also when return value is LOCERR_BUFTOOSMALL, to indicate the required buffer size
**Returns:**
LOCERR_OK on success, LOCERR_OUTOFRANGE when array index is out of range SyncML or LOCERR_xxx error code on other failure

**virtual TSyError sysync::TEngineModuleBase::SetValue (KeyH *aKeyH*, cAppCharP *aValName*, uInt16 *aValType*, cAppPointer *aBuffer*, memSize *aValSize*) [pure virtual]**

Writes a named value in specified format passed in memory buffer.

**Parameters:**
*aKeyH[in]* an open key handle
*aValName[in]* name of the value to write
*aValType[in]* type of value passed in, see VALTYPE_xxxx
*aBuffer[in]* buffer containing the data
*aValSize[in]* size of value. For VALTYPE_TEXT, size can be passed as -1 if string is null terminated
**Returns:**
LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**virtual TSyError sysync::TEngineModuleBase::SetValueByID (KeyH *aKeyH*, sInt32 *aID*, sInt32 *aArrayIndex*, uInt16 *aValType*, cAppPointer *aBuffer*, memSize *aValSize*) [pure virtual]**

Writes a named value in specified format passed in memory buffer.

**Parameters:**
*aKeyH[in]* an open key handle
*aID[in]* ID of the value to read
*aArrayIndex[in]* 0-based array element index for array values.
*aValType[in]* type of value passed in, see VALTYPE_xxxx
*aBuffer[in]* buffer containing the data
*aValSize[in]* size of value. For VALTYPE_TEXT, size can be passed as -1 if string is null terminated
**Returns:**
LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**TSyError sysync::TEngineModuleBase::CloseKeyAndNULL (KeyH & *aKeyH*) `[inline]`**

Closes a key and nulls the handle.

**Parameters:**

*aKeyH[in/out]* an open key handle. Will be set to NULL on exit (to make sure it is not re-used)

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

**TSyError sysync::TEngineModuleBase::CloseSessionAndNULL (SessionH & *aSessionH*) `[inline]`**

Closes a session and nulls the handle.

**Parameters:**

*aSessionH[in]* session handle obtained with OpenSession

**Returns:**

LOCERR_OK on success, SyncML or LOCERR_xxx error code on failure

## 9.3 Settings keys supported in SyncML Client Engine

The settings keys and values accessible trough the OpenKeyXXXX, GetValueXXX, SetValueXXX etc. block of routines (see 5.2.3) may vary depending on the version and type of client library. The standard client library supports the following settings keys and values:

### *9.3.1 Global settings keys - accessed using OpenKeyByPath()*

| | |
|---|---|
| **/engineinfo** | **general info about engine (read-only)** |
| version | - SySync full version string |
| platform | - name of the platform |
| name | - name of the Synthesis SyncML engine product |
| manufacturer | - returns Synthesis AG |
| comment | - returns special release comment string, which might indicate special builds like expiring demo versions etc. |
| variantcode | - returns variant classification code of the engine: 0=unknown, 1=STD, 2=PRO, 3=custom variant, 10=DEMO |
| | |
| **/configvars** | **configuration variables (predefined, volatile when written)** |
| platformname | - name of the current platform |
| platformvers | - version string of the current platform |
| globcfg_path | - global system-wide config path (such as C:\Windows or /etc) |
| loccfg_path | - local config path (such as exedir or user's dir) |
| defout_path | - default path to writable directory to write logs and other output by default |
| temp_path | - path where we can write temp files |
| exedir_path | - path to directory where executable resides |
| userdir_path | - path to the user's home directory for user-visible documents and files |
| appdata_path | - path to the user's preference directory for this application |
| prefs_path | - path to directory where all application prefs reside (not just mine) |
| device_uri | - URI of the device (some unique ID, usually a hardware serial number or network derived system name) |
| device_name | - Name of the device hardware (such as a model name) |
| user_name | - name of the currently logged-in user on the host platform |
| conferrpath | - not defined by default, can be set to a file path where XML config parsing error messages will be written to. Can be set to "console" to output XML config parsing errors to the system stdout path (if such a path exists on the platform) |
| | |
| custommanufacturer<br>custommodel<br>customhardwareversion<br>customsoftwareversion<br>customdevicetype<br>customdeviceid | overrides engine's information with client specfic information (will be visible in log file and syncml messages) |
| | |
| xxxx | - user-defined variables (can also set to override default value of one of the above) |

| | |
|---|---|
| **/licensing** | **license (volatile, text/code must be set every time app ist started)** |
| licensetext | - license text |
| licensecode | - Writeonly: license code (setting it will recalculate all the following status variables) |
| regStatus | - Readonly: TSyError status code of currently set license |
| regOK | - Readonly: if true, license is ok |
| productCode | - Readonly: product code from license |
| productFlags | - Readonly: product flags from license |
| quantity | - Readonly: licensed quantity |
| licenseType | - Readonly: license type |
| daysleft | - Readonly: number of days left of expiring license or demo mode (-1 = not expiring) |
| **/profiles** | **Client settings profiles (persistent)** |
| settingsstatus | - TSyError status of the settings. MUST BE CALLED AT LEAST ONCE before opening subkeys |
| overwrite | - (volatile) boolean flag. In case opening settings would cause deleting incompatible settings, this is done only if overwrite is set to 1. |
| provisioningstring | - Writeonly: Allows creating and modifying settings using so-called *provisioning strings*. These are strings which can be created using our free ClientConfigurator tool to package settings in a form that can be easily delivered in various forms. One of them is writing them into the *provisioningstring* value. |
| checkForFeature | - Writeonly. When written with a feature code (see APP_FTR_xxx in engine_defs.h), the write operation either succeeds (=feature available) or returns 204/ DB_NoContent (=feature not available). Note that checkForFeature is also available in the profile to check for features affecting only a specific profile and on the target level for features specific to a single target. |
| **/<profileID>** | **Profile ID (as assigned by engine when profile is created)** |
| profileName | - display name of the profile |
| protocol | - transport protocol: 0=included in URI, 1=http, 2=https, 3=wsp, 4=obex_irda, 5=obex_bt, 6=obex_tcp |
| serverURI | - SyncML Server URI |
| URIpath | - Path element appended to SyncML Server URI (e.g. in case URI is hardcoded) |
| serverUser | - SyncML Server user |
| serverPassword | - SyncML Server password (stored in disguised form) |
| transportUser | - user for login at the transport level (e.g. HTTP auth) |
| transportPassword | - password for transport level login (stored in disguised form) |
| socksHost | - SOCKS proxy address |
| proxyHost | - HTTP proxy address |
| proxyUser | - user for login at the proxy |
| proxyPassword | - password for proxy login (stored in disguised form) |
| encoding | - SyncML encoding (1=WBXML, 2=XML - note that not some client builds only support WBXML) |
| syncmlvers | - SyncML version to use to start session (0=automatic, 1=1.0, 2=1.1, 3=1.2) |
| useProxy | - If set to 1, this indicates that configured proxy server(s) should be used |

| | |
|---|---|
| useConnectionProxy | - if set to 1, this indicates that OS-defined, connection specific proxies should be used |
| timedSyncMobile | - Number of minutes for mobile timed autosync (0=none) |
| timedSyncCradled | - Number of minutes for cradled timed autosync (0=none) |
| dangerFlags | - Readonly: returns the "danger" status of the next sync, i.e. indication when either server (DANGERFLAG_WILLZAPSERVER) or client (DANGERFLAG_WILLZAPCLIENT) side data will be cleared completely and replaced with the other side's content in any of the datastores currently enabled for sync. Client implementations should check these before starting a sync and warn users appropriately. Note that each target has also a *dangerFlag* which can be queried to get each datastore's flags separately. |
| checkForFeature | - Writeonly. When written with a feature code (see APP_FTR_xxx in engine_defs.h), the write operation either succeeds (=feature available) or returns 204/ DB_NoContent (=feature not available). Note that *checkForFeature* is also available in each target to check for features specific to one target and on the profiles container level for global-level features. |
| checkForReadOnly | - Writeonly. When written with a readonly flag value (see RDONLY_xxx in engine_defs.h), the write operation either succeeds (=queried settings should be made readonly in the UI) or returns 204/ DB_NoContent (=queried settings should be editable in the UI). Note that actual profile and target fields may still be technically writable using SetValue() – *checkForReadOnly* is indended to give the UI implementation the needed information to make some fields not editable for the end user. *checkForReadOnly* is also available in each target to check for readonly fields at the target level. |
| readOnlyFlags | - These flags can be set (usually by provisioning, see "provisioningstring") to make certain aspects of a settings profile read-only. See RDONLY_xxx constants in engine_defs.h. Note that these flags don't actually prohibit writing to settings fields, but should be queried by UI code to using *checkForReadOnly* (see above). UI code should never check *readOnlyFlags* directly, because depending on the engine build some readonly conditions might exist without the corresponding flag explicitly set in *readOnlyFlags*. |
| transpFlags | - 32 bit Flagword reserved for transport related settings flags. |
| profileFlags | - 32 bit Flagword reserved for general profile related settings flags. |
| profileExtra1 | - 32 bit Integer reserved for general profile related settings value. |
| profileExtra2 | - 32 bit Integer reserved for general profile related settings value. |
| profileData | - 256 bytes general purpose BLOB reserved for general profile related persistent storage. |

**/autosynclevels**

| | |
|---|---|
| **/<id>** | **Autosync level ID, 0..2, 0=first priority, 2=least priority** |
| mode | - Autosync mode for this level (0=IPP, 1=timed, 2=off, 3=server alerted) |
| startDayTime | - minute of the day when autosync starts in this level |
| endDayTime | - minute of the day when autosync ends in this level |
| weekdayMask | - weekdays where autosync is enabled in this level (Bit 0=Sun, 1=Mon .. 6=Sat |
| chargeLevel | - percentage of battery charge needed to enable autosync (0..100, 100=with AC supply only) |

| | |
|---|---|
| memLevel | - percentage of memory free needed to enable autosync (0..100) |
| flags | - flags reserved for future use |
| **/targets** | **Targets (databases available for sync in this profile)** |
| **/<targetID>** | **Target ID is the <dbtypeid> as defined in the <datastore> config** |
| enabled | - if set to 1, this datastore will be included in next sync |
| forceslow | - if set to 1, next sync will be a slow sync |
| syncmode | - sync mode: 0=twoway, 1=from server only, 2=from client only |
| limit1 | - sync range limit (such as number of days in the past, depends on datastore) |
| limit2 | - sync range limit (such as number of days in the future, depends on datastore) |
| extras | - flags for sync range limist (depends on datastore) |
| localpath | - local database path (if any), to differentiate multiple instances of the same database type |
| remotepath | - remote (server) database path |
| localcontainer | - local container name, if any (usage depends on datastore implementation) |
| dbname | - Readonly: name of the related <datastore> (in the XML config) |
| lastSync | - Readonly: time of last successful sync |
| lastToRemoteSync | - Readonly: time of last sync that sent data to the remote party (server) |
| dangerFlags | - Readonly: returns the "danger" status of the next sync, i.e. indication when either server (DANGERFLAG_WILLZAPSERVER) or client (DANGERFLAG_WILLZAPCLIENT) side data will be cleared completely and replaced with the other side's content. Client implementations should check these before starting a sync and warn users appropriately. Note that the profile has also a *dangerFlag* which represents all target's *dangerFlag*s combined. |
| checkForFeature | - Writeonly. When written with a feature code (see APP_FTR_xxx in engine_defs.h), the write operation either succeeds (=feature available) or returns 204/ DB_NoContent (=feature not available). Note that checkForFeature is also available in the profile to check for features affecting all targets and on the profiles level for global-level features. |
| checkForReadOnly | - Writeonly. When written with a readonly flag value (see RDONLY_xxx in engine_defs.h), the write operation either succeeds (=queried settings should be made readonly in the UI) or returns 204/ DB_NoContent (=queried settings should be editable in the UI). Note that actual profile and target fields may still be technically writable using SetValue() – *checkForReadOnly* is indended to give the UI implementation the needed information to make some fields not editable for the end user. *checkForReadOnly* is also available at the profile level. |
| isAvailable | - Readonly. Returns non-zero if the datastore is available for being used (vs. only implemented, but currently blocked, e.g. because the server side does not support the type). The UI implementation should visually show the datastore related UI in a disabled state or completely hide it when isAvailable returns zero. |
| dispName | - Readonly. Returns the display name of the datastore as specified in the XML config with <displayname>. If <displayname> is not set, the technical name of the datastore is returned. |

| | |
|---|---|
| lastSyncIdentifer | - Readonly: Returns the identifier used by the datastore implementation (possibly a plugin) to identify the time of last sync. |
| remoteDispName | - Readonly: Returns the display name of the datastore as transmitted in the remote party's devInf (if at all contained in the devInf). |
| remoteFilters | - Filter expression to be passed to server (in TAF/CGI syntax format). |
| localFilters | - Filter expression to be used locally to synchonize only a subset of the local data set (in TAF/CGI syntax format). Not active yet, reserved for future use. |
| filterCapDesc | - Readonly. Reserved for future use (will contain a description of filter capabilities of the server for creating UI like popup menus for filter creation). |

### 9.3.2 Session local settings/values, accessed using OpenSessionKey()

| **Session key** | **unnamed implicit per-session key obtained by OpenSessionKey()** |
|---|---|
| connectURI | - URI to use to connect to SyncML server. Note that this might be different from the original Server URI in profile's "serverURI" as the SyncML server might request sending requests to another URI during a sync session. |
| connectHost | - This is the host (server address) part of the connectURI. This is what is normally required to create a connection at the network level. |
| connectDoc | - This is the document part of the connect URI, which is normally required to prepare a HTTP POST request. |
| contenttype | - content type string to use for the HTTP "Content-Type:" header. |
| localSessionID | - local identification string of the current sync session. |
| sessionPassword | - this is a write-only value. It can be used to provide the session password from a secure storage (like Mac OS X keychain) rather than actually storing it in the profile settings (from where it could be extracted by unauthorized parties).<br>To provide the session password via this value, it must be set immediately after the initial STEPCMD_CLIENTSTART or STEPCMD_CLIENTAUTOSTART has been successfully executed. |
| **/sessionvars** | **Session context script variables (for PRO engines with scripting only)** |
| <varname> | - access (read and write) to any script variable defined in *session context* scripts (like <sessioninitscript>). This is useful to pass extra data back and forth between engine and database plugins or UI. |
| **/profile** | **Access to current session's profile record. See /<profileID> above for description of profile values and subkeys.**<br>This is useful for example for database plugins to make use of profile flags and settings configured for the current session. |

# 10. Error codes

This section lists the error codes that can occur (normally visible in the logs or on the console).

## 10.1 SyncML Status Codes

These codes are defined by the SyncML standard. For details, see http://www.openmobilealliance.org/release_program/ds_v12.html. Note that this list is not complete, but only contains the codes that are important for the SyncML engine.

| | |
|---|---|
| **0** | No error |
| **101** | Server is busy (session limit reached) |
| **200** | OK, successful operation |
| **201** | Item added |
| **204** | no content / end of file / end of iteration / empty/NULL value |
| **207** | Conflict resolved with merge |
| **208** | Conflict resolved - client wins |
| **209** | Conflict resolved by duplicating item |
| **210** | Deleted without archive |
| **211** | Item not deleted |
| **212** | Authentication accepted for entire session |
| **213** | Chunked item accepted and buffered (this status is sent for each non-final part of a data item that has been split across multiple SyncML messages) |
| **400** | Bad request |
| **401** | Unauthorized (bad credentials) |
| **403** | Forbidden (e.g. attempt to write to a read-only database) |
| **404** | Object not found |
| **405** | Command not allowed |
| **406** | Optional feature not supported |
| **407** | Authentication required (no credentials found) |
| **408** | Timeout |
| **409** | Conflict, operation failed |
| **410** | Gone, requested object not here any more |
| **412** | Incomplete command |
| **415** | Unsupported media type or format |
| **418** | Object already exists |
| **419** | Conflict resolved with server data |
| **420** | Device full |
| **500** | Command failed |
| **501** | Command not implemented |
| **503** | Service unavailable |
| **505** | DTD version not supported |
| **508** | Slow sync required |
| **509** | Authentication required |
| **510** | Database error |

| | |
|---|---|
| **511** | Server error |
| **512** | Synchronisation failed |
| **513** | SyncML Version not supported |
| **514** | Cancelled |

## 10.2 Internal Error Codes

| | |
|---|---|
| **0** | No error |
| **10000..10999** | These have the same meaning as the SyncML Status Codes (see 10.1), but they are offset by 10000 to make clear that they were generated internally, and not sent or received via SyncML. |
| **20001** | Bad or unknown transport protocol |
| **20002** | Fatal problem with SyncML encoder/decoder |
| **20003** | Cannot open communication |
| **20004** | Cannot send data |
| **20005** | Cannot receive data |
| **20006** | Bad content type (message received with an unknown MIME-type) |
| **20007** | Error processing incoming SyncML message (for example invalid XML or WBXML formatting) |
| **20008** | Cannot close communication |
| **20009** | Transport layer authorisation (e.g. HTTP auth) failed |
| **20010** | Error parsing XML config file |
| **20011** | Error reading config file |
| **20012** | No configuration found at all, or not enough for requested operation (client) |
| **20013** | Config file could not be found |
| **20014** | License expired or no license found |
| **20015** | Wrong usage |
| **20016** | Bad handle |
| **20017** | Session aborted by user |
| **20018** | Invalid license |
| **20019** | Limited trial version |
| **20020** | Connection timeout |
| **20021** | Connection SSL certificate expired |
| **20022** | Connection SSL certificate invalid |
| **20023** | incomplete sync session (some datastores failed, some completed) |
| **20025** | Out of memory |
| **20026** | Connection impossible (e.g. no network available) |
| **20027** | Establishing connection failed (e.g. network layer login failure) |
| **20028** | element is already installed |
| **20029** | this build is too new for this license (need upgrading license) |
| **20030** | function not implemented |
| **20031** | this license code is valid, but not for this product (e.g. STD license used in PRO product, or client license in server product) |
| **20032** | Explicitly suspended by user |
| **20033** | this build is too old for this SDK/plugin |

| | |
|---|---|
| **20034** | unknown subsystem |
| **20036** | local datastore not ready |
| **20037** | session should be restarted from scratch |
| **20038** | internal pipe communication problem |
| **20039** | buffer too small for requested value |
| **20040** | value truncated to fit into field |
| **20041** | bad parameter |
| **20042** | out of range |
| **20043** | external transport failure (no details known in engine) |
| **20044** | class not registered |
| **20045** | interface not registered |
| **20046** | bad URL |
| **20047** | server not found |
| **20048** | Synthesis server not reachable (sending log files, apply for a temporary license) |
| **20049** | empty answer received |
| **20500..20599** | These represent SIG_xxx codes in Linux versions of the server.<br>Unexpected SIG_xxx will generate a error code of 20500+signal_code. |
| **20998** | Internal unkown exception |
| **20999** | Unknown error |
| **21000...21999** | Database plugin  module specific error codes |