ENOVIA DesignSync
C-API Programmer's Guide

3DEXPERIENCE 2022

Table of Contents

Overview 1

DesignSync C API Programmer's Guide 1

Who Should Read this Guide? 1

About This Guide 1

DesignSync C API 2

Background 2

C API Function Prototype 7

About C API Arguments 9

Pointer to an Array 10

Memory Management 11

Standard Programming Tasks and Practices 13

Using C API Functions 17

How C API Functions Map to DesignSync Commands 17

C API Functions 17

Sample C API Application Source Code 19

Function Summaries 29

How C API Functions Map to DesignSync Commands 29

C API Functions 29

sapiAddStrings() 31

sapiCancel() 35

sapiCheckin() 40

sapiCheckout() 45

sapiExecuteCmd() 49
sapiFreeStrArray() 51
sapiFreeString() 54
sapiGetTclInterp() 56
sapiInitialize() 57
sapiLog() 58
sapiMakeBranch() 59
sapiMakeStrArray() 62
sapiMkFolder() 74
sapiPopulate() 76
sapiRemoteStcl() 79
sapiRetire() 82
sapiRmfile() 84
sapiRmfolder() 88
sapiRmvault() 94
sapiRmversion() 96
sapiRun() 99
sapiSetmirror() 102
sapiSetowner() 102
sapiSetvault() 103
sapiShutdown() 106
sapiSwitchlocker() 106
sapiSyncinfo() 109

sapiTag() 112
sapiUnlock() 112
sapiUrlBranchid() 116
sapiUrlConfigs() 119
sapiUrlContainer() 122
sapiUrlContents() 125
sapiUrlExists() 128
sapiUrlFetchedstate() 132
sapiUrlFetchtime() 136
sapiUrlGetprop() 138
sapiUrlInconflict() 143
sapiUrlLeaf() 146
sapiUrlLocktime() 148
sapiUrlMembers() 151
sapiUrlMirror() 153
sapiUrlModified() 157
sapiUrlOwner() 160
sapiUrlPath() 162
sapiUrlProjects() 165
sapiUrlProperties() 167
sapiUrlRegistered() 176
sapiUrlRelations() 179
sapiUrlResolvetag() 182

sapiUrlRetired() ... 185
sapiUrlServers() ... 187
sapiUrlSetprop() ... 189
sapiUrlSyslock() ... 190
sapiUrlTags() ... 193
sapiUrlVault() ... 196
sapiUrlVaultmirror() ... 199
sapiUrlVersionid() ... 203
sapiUrlVersions() ... 205
Getting Assistance ... 209
Using Help ... 209
Getting a Printable Version of Help ... 209
Contacting ENOVIA ... 210
Index ... 211

Overview

DesignSync C API Programmer's Guide

This guide documents the ENOVIA Synchronicity DesignSync® Data Manager C Application Programmer Interface (C API). It also describes How C API Functions Map to DesignSync Commands In addition, this guide introduces you to the Standard Programming Tasks and Practices associated with creating C API applications that customize DesignSync functionality.

Commented C-language source code examples are provided throughout this guide to demonstrate these coding practices and suggest efficient programming styles. These examples are intended for illustration only. A tested, fully functioning example of a C-API program is available in the directory `$SYNC_DIR/scapi`.

Who Should Read this Guide?

This information is intended for C/C++ programmers and other developers who have recent experience working with high-level languages in a distributed client-server environment. Familiarity with DesignSync or related revision control software and Web-based technologies is assumed, as is significant hands-on experience with UNIX, Windows, and interpreted scripting languages.

About This Guide

This guide provides the following information about the DesignSync C API:

- DesignSync C API
  This overview summarizes the background, structure, and purpose of the C API.
- C API Function Prototype
  More than 90 percent of all C API calls are based on a single function prototype.
- Standard Programming Tasks and Practices
  This section contains a brief programming "cookbook" that lists specific programming steps and introduces memory-management strategies
- How C API Functions Map to DesignSync Commands
  This list of C API function calls provides useful links to syntax statements, argument definitions, command descriptions with usage examples, return values, error handling, and associated DesignSync commands. The mapping information highlights the direct relationship between C API functions and DesignSync commands.
- Sample C API Application Source Code
  This heavily commented source code examples demonstrate how to use the C API to implement two sample applications.
- Function Summaries
  This list of C API function calls in this section provides syntax statements,

argument definitions, command descriptions with return values, error handling, and provides links to associated DesignSync commands, when applicable.

Note: References from the *ENOVIA Synchronicity DesignSync Data Manager C-API Programmer's Guide* to the *ENOVIA Synchronicity Command Reference* guide always link to the ALL version of the guide, which contain information about all working methodologies for DesignSync. For more information about the available working methodologies, see ENOVIA Synchronicity Command Reference.

DesignSync C API

DesignSync features a formalized C-language layer. This architectural enhancement provides code-level support for the C-language API (C API) described in this documentation. The C API enables you to construct C/C++ applications capable of dynamically customizing and extending DesignSync functionality directly from a client-side application.

Background

In the context of this document, the programming terms in **boldface** are defined as indicated in this section. If a definition seems unfamiliar, please review the associated description before continuing.

SOM Objects and web Objects

Every object in the DesignSync user environment is a **DesignSync Object Model (SOM)** object. SOM objects are core DesignSync objects capable of manipulating URL-addressable **web objects** (files, directories, vaults, versions, and so on) directly from a client application or from a SyncServer. In the DesignSync environment, the following URL prefixes help identify key differences among web objects:

URL Prefix	Description
`file:///`	Identifies a web object that resides on the client side. The URL contains "local" (client-side) path information.
`sync:///host:port/`	Identifies a web object that resides on the client side but is managed by a SyncServer.
`sync://host:port/`	Identifies a web object that resides on the SyncServer that manages it. For example, a server-side DesignSync script would access a server-side web object using this prefix.

DesignSync Commands and C API Function Calls

The DesignSync Graphical User Interface (GUI) the DesignSync shell command (dss), concurrent dss (dssc), the Synchronicity tcl shell (stcl), and concurrent stcl (stclc) all enable you to access the DesignSync **command set**. These are all client applications.

The DesignSync C API represents an alternative interface whose function calls map directly to individual commands in the DesignSync command set. The program created using the C API is also a client application. The `scapi.h` header contains the C API library function declarations and a list of `#defines` covering most options and flags. This header file and a working example of how to use the C API are in the directory `$SYNC_DIR/scapi`.

The following table compares these related interfaces by comparing and contrasting their arguments:

Interface Name	**Description of Arguments**
DesignSync command set	Command arguments are called **options**. Most options have a **name** and an associated **value**. Options that have a name but no associated value are known as **flags**. The functionality associated with these commands and their supported options are fully documented in How C API Functions Map to DesignSync Commands.

DesignSync C API	All C API functions that map to DesignSync commands require the following arguments: • `extra` Reserved for future use. For now, you specify a NULL, which the `scapi.h` header file defines as a placeholder. • `inputOptions` Even when the command does not have options or flags, each C API function that maps to a DesignSync command requires the `inputOptions` array. This is a `char**` pointer to a NULL-terminated array of name, value strings. Each name string in the array specifies a single command option or flag. For an option, the value> string specifies the associated value of the option. For a flag, the value string is represented by an empty string (a pair of double quotes, `""`) because flags do not have an associated value. If there are no options or flags to specify for this command (or operation), then the NULL keyword, representing a null pointer, must be supplied. • `results` A `char***` pointer to array of pointers to the results of the associated command. If the command fails, an error message is returned to the caller in the `results` array. See About_C_API_Arguments for more information. In addition, functions that map to DesignSync commands usually work on specific object types, such as a single web object (`webObject`), multiple web objects (`webObjectList`), files, folders, versions, and so forth. These commands take an additional `char*` pointer or `char**` pointer to the object(s) to be operated on. See C API Function Prototype for the different types of arguments that might be required. Some C API function calls require more than one argument of this type; other functions do not require you to specify any objects. For more information about these and other arguments, see How C API Functions Map to DesignSync Commands. See Memory Management for information about creating,

	releasing, and adding to an array of strings.

EXAMPLES:

The DesignSync `ci` command, (which checks in files) maps to `sapiCheckin()`, which is a C API function. This function requires you to specify the following four arguments:

- `extra`
  This argument is reserved for future use. For now, specify a NULL, which is defined in the `scapi.h` include file as a placeholder.

- `webObjectList`
  A `char**` pointer to an array of objects representing all web objects being checked in.
- `inputOptions`
  A `char**` pointer to a NULL-terminated array of name, value strings representing the DesignSync command options or flags that must be applied to the web objects being checked in.
- `results`
  A `char***` pointer to an array of pointers to the results of each `ci` command option.

Purpose

DesignSync currently supports the following interfaces to enable you to use, customize, and extend DesignSync functionality:

- The DesSync graphical user interface (GUI).
- The DesignSync shell (`dss`) and the concurrent version of `dss` (`dssc`), which does not launch a DesignSync daemon (`syncd`) process.
- The Synchronicity Tcl shell (`stcl`) and the concurrent version of `stcl` (`stclc`), which does not launch a DesignSync daemon (`syncd`) process.

Most DesignSync commands associated with these interfaces can, like the DesignSync C API, run from any DesignSync client. However, the C API is not intended to replace the DesignSync commands. Instead, the C API provides C/C++ programmers with a familiar compiled alternative that supports statically linked libraries and dynamically linked libraries (DLLs).

Note: Some familiarity with the DesignSync command set is a prerequisite for using the C API because, with rare exception, each C API call:

- Maps directly to a DesignSync command whose name and functionality it duplicates.
- Takes arguments that are pointers to:
  - The object(s) processed by the mapped DesignSync command.
  - The command options and flags of the mapped DesignSync command.

The close relationship between C API calls and DesignSync commands creates a high degree of syntactic overlap. For a complete description of the DesignSync commands, see the ENOVIA Synchronicity Command Reference.

One Difference Worth Considering

Unlike DesignSync scripts, C API applications are not interpreted by a command shell. Instead, like all C programs, they are maintained in source code files that must be

compiled into executable binaries after every code update and linked to appropriate function libraries before they can be run.

C API Function Prototype

Every C API function has a prototype whose syntax is similar to, though not always identical to, the statement shown here. You use this syntax to declare each C API function in your application. All C API function definitions are stored externally (in the `scapi.h` header file):

Syntax

```
extern HRESULT sapiFunctionName  (void* extra,
                                  char** webObjectList,
                                  char** inputOptions,
                                  char*** results);
```

Arguments

`extra`	Reserved for future use. Specify a NULL, which the `scapi.h` header file defines as a placeholder.

`webObjectList` A pointer to a string (or NULL-terminated array of strings). Each string refers to the web object to be processed as either a relative path, absolute path, a URL, a file name, a folder name, or sometimes even "." to represent the current folder.

Some C API functions substitute one or more of the following object lists for the generic `webObjectList`:

- `fileList`
- `folderList`
- `vaultList`
- `versionList`

When the equivalent DesignSync command takes a single web object, its C API function requires you to specify a `char*` pointer argument rather than a `char** webObjectList` pointer-to-array argument. The names of these arguments indicate that they specify pointers to single items rather than arrays. The following arguments are the most commonly specified single-item C API function arguments:

- `webObject`
- `object`
- `mirrorDir`
- `localDir`
- `owner`
- `vault`
- `vaultDirURL`
- `filename`
- `containerObj`
- `collectionObj`
- `logFileName`
- `argList`
- `branchTag`

Some C API function calls require more than one `webObjectList` argument; other functions do not require you to identify any web objects.

The specified arguments are analogous to those specified for the corresponding DesignSync commands.

Note: See Memory Management for information about creating, releasing, and adding to an array of web object (URL) strings.

`inputOptions` A NULL-terminated array of name, value strings, each specifying a single command option or flag. Each array of strings is terminated by a single NULL entry. If the function requires the `inputOptions` argument but it is not needed for the operation, supply the `NULL` keyword, representing a null pointer.

Flags also are specified as name, value strings. The `value` is represented by an empty string (a pair of double quotes, `""`) because flags do not have an associated `value`.

EXAMPLE:

```
char** inputOptions = {
"-option1", "value for option1 switch",
"-flagX", "",
"-flagY", "",
NULL
}
```

A list of `#defines` in the `scapi.h` header file covers every possible option and flag.

Note: See Memory Management for information about C API functions that simplify the process of creating, releasing, and adding to these arrays.

`results` Pointer to array of pointers to the results. If the command fails, an error message is returned to the caller in the `results` array. See About C API Arguments for more information.

Returns

`HRESULT` Each C API function returns a long integer, `HRESULT`, for the error code. Zero (`0`) indicates success; a negative integer indicates failure. Use the `SUCCEEDED` and `FAILED` macros to test for success and failure.

The `scapi.h` header file and a working example of how to use the C API are in the directory `$SYNC_DIR/scapi`.

About C API Arguments

The C API function prototype specifies four arguments. Three of these arguments are required for all C API functions that map to DesignSync commands. However, a few C API functions that do not map to a DesignSync command (for example, the memory

management convenience functions) require different arguments. Other functions require more than four arguments because the commands they map to require the additional arguments.

EXAMPLES:

- `sapiPopulate()`
  This function corresponds to the DesignSync `populate` command. It requires only the three standard arguments required by every command that maps to a function: a `void*` placeholder argument (`extra`), a `char**` argument (`inputOptions`), and a standard `results` argument.
- `sapiCheckout()`
  This function corresponds to the DesignSync `co` command. It has the four standard C API arguments.
- `sapiSetmirror()`
  This function corresponds to the DesignSync `setmirror` command. It associates a mirror directory with a local working directory. The following syntax statement describes its five required arguments:

```
extern HRESULT sapiSetMirror(void* extra,
                                        char* mirrorDir,
                                        char* localDir,
                                        char**
inputOptions,
                                        char*** results);
/*
// extra is reserved for future use. Specify a NULL.
// mirrorDir is a pointer to a mirror directory.
// localDir is a pointer to the local directory associated
//   with the mirror.
// inputOptions is a NULL-terminated array of name, value
strings.
// results is a pointer to an array of pointers to the
results.
*/
```

Pointer to an Array

The C/C++ mechanism "pointer to an array" is widely employed in the DesignSync C API for passing input options. It eliminates:

- Recompilation of the client programs whenever a command adds options
- The need for footprint changes

The results Argument

A significant benefit of returning a pointer to an array of strings indicating the results of each command option is that the function prototype can accommodate future increases in the amount of returned data without itself requiring alteration.

The `results` array returned is a NULL-terminated array of strings and is analogous to the `RETURN VALUE` described (for stcl/stclc mode) in the ENOVIA Synchronicity Command Reference. If only a single value is returned, it is returned in the first element of the `results` array. If an error is thrown, the error string is returned in the first element of the `results` array.

EXAMPLE:

The C API `sapiCheckin()` function maps to the DesignSync `ci` command. A **results** array for `sapiCheckin()` looks like this, where `n` represents the total number of objects that succeeded and failed:

```
char** results = {
"Objects succeeded (n)", // Successes
"Objects failed (n)", // Failures
NULL
}
```

Note: In this example **results** is a `char**` even though the C API Function Prototype casts it as a `char***`. Although you initially cast `results` as a `char**` (a pointer to an array of strings), you must pass the C API function a pointer to the `char**` pointer.

The `sapiUrlExists()` function is equivalent to the DesignSync `urlexists` command. A `results` array for `sapiUrlExists()` looks like this:

```
char** results = {
"1", // the url exists
NULL
}
```

Note: The outgoing `results` array is constructed by the C API function and passed back in the supplied pointer. When no longer needed, it must be freed using the appropriate C API function, because the memory allocation occurred within the C API dynamic-link library (DLL). See Memory Management for information about releasing this array.

See Memory Management for information about C API functions that simplify the process of creating, releasing, and adding to these arrays

Memory Management

Like most C-language APIs, the DesignSync C API holds the programmer responsible for general memory management. If you allocate memory, you are responsible for freeing that memory. However, the C API does provide some useful memory allocation functions to help you manage memory. These functions are defined and described in the following table.

CAUTION: If you use C API memory allocation functions, you must also use the associated deallocation functions shown in this section to free allocated memory because all memory allocation occurs within the C API dynamic library.

The C API provides the following functions to simplify the allocation and deallocation of string array memory:

Function Prototype	Description
`extern HRESULT sapiMakeStrArray ( char*** strArray, ...);`	Given strings as arguments, this C API function creates and returns a pointer to an array of strings in the outgoing `strArray` argument. The ellipsis (`...`) denotes a NULL-terminated variable argument list. In your source code, define `strArray` as a `char**` pointer to the specified array and pass a pointer to `strArray` (in other words, a `char***` pointer) to this function. The required arguments are assigned names that have meaning in the context of this function. If this function is called with no string arguments, it creates an array of `char*` pointers with no strings inserted.
`extern HRESULT sapiFreeStrArray( char** strArray);`	Frees all the strings in the array created by `sapiMakeStrArray()` and releases the array itself. This function should also be used to free the `results` array.
`extern HRESULT sapiAddStrings( char** strArray, ...);`	Given `strArray`, a string array previously created with the `sapiMakeStrArray()` function, this function adds to the specified array the strings you specify in place of the ellipsis (`...`), which denote a NULL-terminated variable argument list. If you use this function to add strings to the `inputOptions` array remember that a name, value pair must be added for each

	option and flag. However each flag's value is always represented by an empty string (a pair of double quotes, `""`) because although flags have names, they do not have associated values.
`extern HRESULT sapiFreeString( char* resultstring);`	Frees the resulting string returned by the `sapiExecuteCmd()` function.

Standard Programming Tasks and Practices

The programming tasks presented in this section are applicable to all C API programs. These tasks are summarized here in numbered steps to suggest the flow of programming tasks common to C API programs:

1. Include the following ANSI standard C header files:

   ```
   #include <stdio.h>
   #include <string.h>
   ```

2. Include the following C API library header file, which contains declarations for all the DesignSync C API functions, related `#defines`, the `SUCCEEDED` and `FAILED` macros, and other required code elements:

   ```
   #include scapi.h
   ```

   This header file and a working example of how to use the C API are in the directory `$SYNC_DIR/scapi`.

3. Declare all required variables. For example, in the Sample C API Application Source Code, the following variables are declared:

   ```
      /*
      // Declare the following NULL pointers and other
   variables
      // required for this application
   */
         char *ecresult = NULL;
         char **resArray = NULL;
         char **objectList = NULL;
         char **inputOptions = NULL;
         HRESULT hr;
         int i;
         char logdir[128];
   ```

4. Define the `main()` function. The `argc` variable preserves the number of command-line arguments with which your application is invoked. The `argv` variable points to the array of argument strings with which your application is invoked.

```
int main (int argc, char *argv[]){
. . .(Your C API function calls go here). . .
return S_OK;
}
```

5. Call the C API initialization function, passing it `argc` and `argv` so that they are available if required:

```
sapiInitialize(argc, argv);
```

The functions called by `sapiInitialize()` set up a DesignSync session. This process includes setting up the underlying DesignSync Object Model (SOM) context, setting up the tcl interpreter, auto-loading tcl procedures, initializing the registry files, and so on. Without this call, the C API functions do not work.

6. Determine which combination of DesignSync commands and options provide the functionality required by your C API program.
7. Determine which C API functions map to the DesignSync commands and options you identified.
8. Use the mapped C API functions and their required arguments to implement the equivalent functionality. The following C API convenience functions help minimize the code required to create, initialize, and manage string arrays, and to allocate and deallocate required memory:

a. Use the `sapiMakeStrArray()` function to create and initialize each new `webObjectList` array and `inputOptions` array - or any other `char**` array of strings required for your C API program. Every `inputOptions` array you create by calling `sapiMakeStrArray()` must contain a name, value pair for each DesignSync command option and a name, `""` (empty string) for each flag.

b. Once the `webObjectList` array or the `inputOptions` array is initialized, add strings by calling `sapiAddStrings()`.

c. After an array has been initialized with the `sapiMakeStrArray()` function and has served its intended purpose, you have two options: free the string by calling `sapiFreeStrArray()` or pass to the function the same `char***` pointer to the `sapiMakeStrArray()` function that you used previously. The existing, currently allocated array will be reused and extended if necessary.

Programming Tip: DesignSync includes a table of addresses for all C API-allocated arrays. This table also contains the number of pointers allocated for each array. To save an array - including an array created with

`sapiMakeStrArray()` or the array of results specified for all of the C API functions - you must pass in a new pointer.

9. At the end of the C API program, release the SOM context by calling the following function:

   ```
   sapiShutdown();
   ```

See Sample C API Application Source Code for illustrations of how these standard programming tasks and practices used in a sample program. In addition, a more extensive example is included in directory `$SYNC_DIR/scapi`.

Using C API Functions

How C API Functions Map to DesignSync Commands

This section describes each C API function and, if applicable, the equivalent DesignSync command. The hyperlinked, alphabetized table of C API Functions lets you quickly locate information about each C API function, including its syntax, arguments, return values, and coding examples. The table also identifies the DesignSync command that each function maps to. A full description of the DesignSync commands is available in the ENOVIA Synchronicity Command Reference.

The `results` array returned is a NULL-terminated array of strings and is analogous to the `RETURN VALUE` shown (for stcl/stclc mode) in the ENOVIA Synchronicity Command Reference. If only a single value is returned, it is returned in the first element of the results array. If an error is thrown, the error string is returned in the first element of the results array. See About C API Arguments for more information.

C API Functions

DesignSync C API functions can be divided into the following categories:

- **Functions that map to DesignSync commands**
  Most C API functions fall into this category
- **Functions that do not map to DesignSync commands**
  These functions are "convenience" functions that:
  - Assist you with memory management
  - Establish an appropriate run-time context for your C API application
  - Ensure the graceful shutdown of your C API application.
  - Provide access to the Tcl interpreter loaded with all DesignSync and tcl commands. This access is useful for integration with third-party tools that require access to the tcl interpreter.

Note: The C API functions can access only client-side functionality and commands because the compiled application is a client program, similar to the DesignSync GUI, stcl/stclc and dss/dssc.

All of these DesignSync C API functions are listed alphabetically in the following table. Functions that map directly to a DesignSync command display that command inside a set of square brackets ([ ]) The convenience functions that do not map to DesignSync commands display the word **none** in square brackets.

Each function name begins with the same identifying prefix (`sapi`) and follows standard upper/lower case C/C++ naming conventions. The majority of the functions are based on a standard C API Function Prototype and take many, or all, of the arguments it defines.

sapiAddStrings() [none]	sapiCancel() [cancel]	sapiCheckin() [ci]	sapiCheckout() [co]
sapiExecuteCmd() [none]	sapiFreeStrArray() [none]	sapiFreeString() [none]	sapiInitialize() [none]
sapiLog() [log]	sapiMakeBranch() [mkbranch]	sapiMakeStrArray() [none]	sapiMkFolder() [mkfolder]
sapiPopulate() [populate]	sapiRemoteStcl() [rstcl]	sapiRetire() [retire]	sapiRmfile() [rmfile]
sapiRmfolder() [rmfolder]	sapiRmvault() [rmvault]	sapiRmversion() [rmversion]	sapiRun() [run]
sapiSetmirror() [setmirror]	sapiSetowner() [setowner]	sapiSetvault() [setvault]	sapiSetvaultmirror() [setvaultmirror] (obsolete)
sapiShutdown() [none]	sapiSwitchlocker() [switchlocker]	sapiSyncinfo() [syncinfo]	sapiTag() [tag]
sapiUnlock() [unlock]	sapiUrlBranchid() [url branchid]	sapiUrlConfigs() [url configs]	sapiUrlContainer() [url container]
sapiUrlContents() [url contents]	sapiUrlEnsurepathname() [url ensurepathname] (unsupported)	sapiUrlExists() [url exists]	sapiUrlFetchedstate() [url fetchedstate]
sapiUrlFetchtime() [url fetchtime]	sapiUrlGetprop() [url getprop]	sapiUrlInconflict() [url inconflict]	sapiUrlLeaf() [url leaf]
sapiUrlLocktime() [url locktime]	sapiUrlMembers() [url members]	sapiUrlMirror() [url mirror]	sapiUrlModified() [url modified]
sapiUrlOwner() [url owner]	sapiUrlPath() [url path]	sapiUrlProjects() [url projects]	sapiUrlProperties() [url properties]
sapiUrlRegistered() [url registered]	sapiUrlRelations() [url relations]	sapiUrlResolvetag() [url resolvetag]	sapiUrlRetired() [url retired]
sapiUrlServers() [url servers]	sapiUrlSetprop() [url setprop]	sapiUrlSyslock() [url syslock]	sapiUrlTags() [url tags]
sapiUrlVault() [url vault]	sapiUrlVaultmirror() [url vaultmirror]	sapiUrlVersionid() [url versionid]	sapiUrlVersions() [versions]

Click on a function name to display a summary of the function and the DesignSync command it maps to, including pertinent syntax statements, arguments, return values, error handling information, usage examples, and other useful information.

Sample C API Application Source Code

The following source code is heavily commented. It demonstrates the use of several C API function calls to complete a series of tasks outlined in the comments at the top of the file:

```
/*
////////////////////////////////////////////////////////
// Copyright (c) 1997-2010 Dassault Systemes.  All rights
reserved.
// Use of this source code is restricted to the terms of your
license
// agreement with Dassault Systemes.  Any use, reproduction,
// distribution, copying or re-distribution of this code outside
the scope
// of that agreement is a violation of U.S. and International
Copyright laws.
////////////////////////////////////////////////////////
//
// This sample application is implemented with C API functions
// that accomplish the numbered tasks listed below. Most of
// the functions map directly to DesignSync commands (shown
// in square brackets, "[]," to highlight the specific command
// options). Each task demonstrates the functionality of a
single
// mapped C API function or related set of command options.
// Some memory management convenience functions that manipulate
// string arrays are called as required.
//
// Note:
// The code presented in this sample application also appears
// in usage examples in the summaries of individual C API
functions.
//
//    TASKS:
//
//    1. Set the default directory for the creation of log files
//       for this session and future sessions. Also set logging
output
//       so that commands and results will both be logged.
//       [log -defaultdir home/larry/logs -output]
//
//    2. Specify the log file for the current sync session
```

```
//       as 'pcimaster.log'.
//
//       Specifying a default log dir. and a log file are
mutually
//       exclusive operations; a second log command must be
used:
//       [log pcimaster.log]
//
//    3. Display the current logging state.
//       [log -state]
//
//    4. Check out the latest-greatest version of all project
//       files [co -version "Latest"] except shell script files
//       with an .sh extension.
//        [co -exclude *.sh]
//
//    5. Project files reside in multiple subdirectories on
//       the same branch.
//       [co -recursive]
//
//    6. A comment with spaces ("Updated by G. B. Shaw") must
//       be added to all checked-out files.
//       [co -comment "Updated by G. B. Shaw"]
//
//    7. All checked out files must be readied for editing.
//       [co -lock]
//
//    8. If any files of the same name already exist in the
//       local workspace, overwrite them without exception.
//       [co -force]
//
//    9. Retain the "last modified" timestamp
//       [co -retain]
//
//   10. All project files should adhere to the
//       following naming convention:
//          pcimaster_<filename>.vbh
//
//       Project files that do not fit this naming convention
//       should be tagged for later renaming.
//       [tag "RENAME_THIS_FILE" -recursive -exclude
pcimaster_*.vbh]
//       Note:
//       The specified tag will be applied to the current
version, not
//       the new version created by the checkout-checkin
procedures.
```

```
*/


/*
// The following standard ANSI C header files are always
required:
*/

#include <stdio.h>
#include <string.h>

/*
// Also include the DesignSync C API library header file,
// scapi.h, which contains declarations of all C API functions
// and the SUCCEEDEDand FAILED macros.
*/

#include scapi.h

int main(int argc, char *argv[])
{
/*
// Declare the C API functions used in this application,
beginning with
// sapiInitialize(), ending with sapiShutdown(), and including
the
// necessary memory management convenience functions
//
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);


extern HRESULT sapiCheckin (void* extra,
                            char** webObjectList,
                            char** inputOptions,
                           char*** results);

extern HRESULT sapiCheckout (void* extra,
                             char** webObjectList,
                             char** inputOptions,
                             char*** results);

extern HRESULT sapiLog (void* extra,
                        char* logFileName,
                        char** inputOptions,
                        char*** results);
```

```
extern HRESULT sapiTag (void* extra,
                        char** fileList,
                        char* tag,
                        char** inputOptions,
                        char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();

/*
// Declare required variables. This sample application requires
// a variable named "hr" of type HRESULT, an index integer "i,"
// a 128 character array named "logdir," plus several NULL
pointers
// that will serve as C API function arguments.
//
// Note:
// The char * pointer "ecresult," will point to the results of
the
// sapiExecuteCommand() function). The char ** pointers are
described
// in detail in the the topic "C API Function Prototype."
*/

    HRESULT hr;
    int i;
    char logdir[128];
    char *ecresult = NULL;
    char **resArray = NULL;
    char **objectList = NULL;
    char **inputOptions = NULL;

/*
// Call sapiInitialize() to initialize a SOM context and create
// a session sink (a default set of of function calls that
establishes
// environmental settings).
// Pass in the argc and argv arguments from main() and use the
// FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
```

```
              return 1;
       }
/*
//*********** Start Unique Part of Your Application Here
*************
//
// TASK1: Specify default Logging directory on a UNIX Sync
sever.
*/

strcpy(logdir, "/home/larry/logs");


/*
// Call sapiMakeStringArray() to create (and allocate memory
for)
// the NULL-terminated inputOptions string array.
// [log -defaultdir /home/larry/logs -output]
// Notes:
// 1. The "logdir" string is specified as the value of the
DEFAULTDIR
//    variable.
// 2. An empty string (a pair of double quotes, ("") is
specified as
//    value of the OUTPUT flag.
*/
    hr = sapiMakeStrArray (&inputOptions,
                           DEFAULTDIR, logdir,
                           OUTPUT, "",
                           NULL);
/*
// Test for failure with the FAILED macro.
*/
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Use sapiLog() to specify the log directory and logging
format. Note
// the use of the NULL "placeholder" for the logFileName
argument:
// if you specify -default but do not specify a default log
directory,
// the command defaults to your home directory.
*/
    hr = sapiLog (NULL,
                  NULL,
```

```
                    inputOptions,
                    &resArray);

/*
// Free the inputOptions array and the results array created by
// sapiMakeStrArray().
*/
    hr = sapiFreeStrArray (inputOptions);
    hr = sapiFreeStrArray (resArray);
/*
// Programming tips:
// If you use the "inputOptions" array and/or the "resArray"
(results)
// array again within the same function call, there is no need
to call
// sapiFreeStrArray() between the calls. Each time
sapiMakeStrArray()
// is called and each time the char **resArray pointer is passed
to a
// C API function, the strings previously inserted into these
arrays are
// freed (within the C API) and then reallocated for the new
data that
// must be inserted in them.
//
// These arrays must be freed each time a function is exited.
// If these arrays are not freed, a memory leak results.
//
// One way to avoid this potential problem is to declare the
inputOptions
// and resArray pointer variables as "static" variables so they
don't need
// to be freed, i.e., "static char **inputOptions" and
// "static char **resArray." If this same function is called
again in
// the same application, the C API recognizes the pointers and
frees
// the strings in the arrays before they are re-used, just as
noted
// above when reused within the same function with no freeing of
memory.
*/

/*
// TASK 2: Set the logfile name (the argument is logFileName) to
// "pcimaster.log". This must be done with a second call to
sapiLog()
```

```
// because this log process and the log directory specification
process
// are mutually exclusive at the DesignSync command level.
*/
    hr = sapiLog(NULL,
                "pcimaster.log",
                 NULL,
                 &resArray);

    if( FAILED(hr) ) {
       printf("sapiLog failed\n"); return E_FAIL;
    }
/*
// TASK #3: Display current logging state [log -state].
// Because the -state flag is optional, you can call sapiLog
// as shown below. Test for failure with the FAILED macro as
before.
*/
    hr = sapiLog(NULL, NULL, NULL, &resArray);

    if( FAILED(hr) ) {
      printf("sapiLog failed\n"); return E_FAIL;
    }
/*
// Free memory allocated for resArray as before, testing for
failure.
*/
   hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
      printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// TASKS #4 - #9 are all accomplished with a single DesignSync
checkout
// command modified by the following options and flags:
// co -version "Latest" -exclude *.sh -recursive
// -comment "Updated by G. B. Shaw" -lock -force -retain
//
// Call MakeStrArray()to create (and allocate memory for)
// the required sapiCheckout() ObjectList array. The pattern
that
// specifies the file naming convention is pcimaster_*.vbh,
// as described in TASK #7.
//Test for failure with the FAILED macro.
*/
    hr = sapiMakeStrArray(&objectList,
```

```
                            "pcimaster_*.vbh",
                            NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Call MakeStrArray()to create (and allocate memory for)
// the required sapiCheckout() NULL-terminated inputOptions
array:
// [-version "Latest" -exclude *.sh -recursive
// -comment "Updated by G. B. Shaw" -lock -force -retain].
// Test for failure with the FAILED macro as before.
*/
    hr = sapiMakeStrArray(&inputOptions,
                            VERSION, "Latest",
                            EXCLUDE, "*.sh",
                            RECURSIVE, "",
                            COMMENT, "Updated by G. B. Shaw,
                            LOCK, "",
                            FORCE, "",
                            RETAIN "",
                            NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Use sapiCheckout() to checkout the files whose URLs are in
the ObjectList,
// applying all of the options/flags in the inputOptions array
and collecting
// the results in resArray.
// Test for failure with the FAILED macro as before.
*/
    hr = sapiCheckout(NULL, objectList, inputOptions,
&resArray);

    if( FAILED(hr) ) {
       printf("sapiCheckout failed\n");
        return E_FAIL;
    }
/*
// Free the inputOptions array and the resArray (results array)
you created
// with sapiMakeStrArray(). Test for failure with the FAILED
macro.
*/
    hr = sapiFreeStrArray(inputOptions);
```

```
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// TASK#10:
// Tag (for later review) all project files that do not fit
naming convention.
// [tag RenameFile -recursive -exclude pcimaster_*.vbh]
// Note:
// If you tag a locked file, you are tagging the Original
version;
// you cannot tag the Upcoming version until it has been checked
in.
//
// As before, call MakeStrArray()to create and allocate memory
for
// the required NULL-terminated objectList array (use * to
specify
// "all files") and inputOptions array (which contains the
pattern
// for excluding files that match the specified file naming
convention).
// Test for failure with the FAILED macro.
*/
    hr = sapiMakeStrArray(&objectList, "*", NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiTag(NULL, objectList, RECURSIVE, "",
                                   EXCLUDE, "pcimaster_*.vbh",
                                   NULL,
                                   &resArray);
    if( FAILED(hr) ) {
        printf("sapiTag failed\n"); return E_FAIL;
    }

/*
// Free the inputOptions array and the results array you created
with
// sapiMakeStrArray(). Test for failure with the FAILED macro.
```

```
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

Function Summaries

How C API Functions Map to DesignSync Commands

This section describes each C API function and, if applicable, the equivalent DesignSync command. The hyperlinked, alphabetized table of C API Functions lets you quickly locate information about each C API function, including its syntax, arguments, return values, and coding examples. The table also identifies the DesignSync command that each function maps to. A full description of the DesignSync commands is available in the ENOVIA Synchronicity Command Reference.

The `results` array returned is a NULL-terminated array of strings and is analogous to the `RETURN VALUE` shown (for stcl/stclc mode) in the ENOVIA Synchronicity Command Reference. If only a single value is returned, it is returned in the first element of the results array. If an error is thrown, the error string is returned in the first element of the results array. See About C API Arguments for more information.

C API Functions

DesignSync C API functions can be divided into the following categories:

- **Functions that map to DesignSync commands**
  Most C API functions fall into this category
- **Functions that do not map to DesignSync commands**
  These functions are "convenience" functions that:
  - Assist you with memory management
  - Establish an appropriate run-time context for your C API application
  - Ensure the graceful shutdown of your C API application.
  - Provide access to the Tcl interpreter loaded with all DesignSync and tcl commands. This access is useful for integration with third-party tools that require access to the tcl interpreter.

Note: The C API functions can access only client-side functionality and commands because the compiled application is a client program, similar to the DesignSync GUI, stcl/stclc and dss/dssc.

All of these DesignSync C API functions are listed alphabetically in the following table. Functions that map directly to a DesignSync command display that command inside a set of square brackets ([ ]) The convenience functions that do not map to DesignSync commands display the word **none** in square brackets.

Each function name begins with the same identifying prefix (`sapi`) and follows standard upper/lower case C/C++ naming conventions. The majority of the functions are based on a standard C API Function Prototype and take many, or all, of the arguments it defines.

sapiAddStrings() [none]	sapiCancel() [cancel]	sapiCheckin() [ci]	sapiCheckout() [co]
sapiExecuteCmd() [none]	sapiFreeStrArray() [none]	sapiFreeString() [none]	sapiInitialize() [none]
sapiLog() [log]	sapiMakeBranch() [mkbranch]	sapiMakeStrArray() [none]	sapiMkFolder() [mkfolder]
sapiPopulate() [populate]	sapiRemoteStcl() [rstcl]	sapiRetire() [retire]	sapiRmfile() [rmfile]
sapiRmfolder() [rmfolder]	sapiRmvault() [rmvault]	sapiRmversion() [rmversion]	sapiRun() [run]
sapiSetmirror() [setmirror]	sapiSetowner() [setowner]	sapiSetvault() [setvault]	sapiSetvaultmirror() [setvaultmirror] (obsolete)
sapiShutdown() [none]	sapiSwitchlocker() [switchlocker]	sapiSyncinfo() [syncinfo]	sapiTag() [tag]
sapiUnlock() [unlock]	sapiUrlBranchid() [url branchid]	sapiUrlConfigs() [url configs]	sapiUrlContainer() [url container]
sapiUrlContents() [url contents]	sapiUrlEnsurepathname() [url ensurepathname] (unsupported)	sapiUrlExists() [url exists]	sapiUrlFetchedstate() [url fetchedstate]
sapiUrlFetchtime() [url fetchtime]	sapiUrlGetprop() [url getprop]	sapiUrlInconflict() [url inconflict]	sapiUrlLeaf() [url leaf]
sapiUrlLocktime() [url locktime]	sapiUrlMembers() [url members]	sapiUrlMirror() [url mirror]	sapiUrlModified() [url modified]
sapiUrlOwner() [url owner]	sapiUrlPath() [url path]	sapiUrlProjects() [url projects]	sapiUrlProperties() [url properties]
sapiUrlRegistered() [url registered]	sapiUrlRelations() [url relations]	sapiUrlResolvetag() [url resolvetag]	sapiUrlRetired() [url retired]
sapiUrlServers() [url servers]	sapiUrlSetprop() [url setprop]	sapiUrlSyslock() [url syslock]	sapiUrlTags() [url tags]
sapiUrlVault() [url vault]	sapiUrlVaultmirror() [url vaultmirror]	sapiUrlVersionid() [url versionid]	sapiUrlVersions() [versions]

Click on a function name to display a summary of the function and the DesignSync command it maps to, including pertinent syntax statements, arguments, return values, error handling information, usage examples, and other useful information.

sapiAddStrings()

This C API function adds specified strings to an array previously created with `sapiMakeStrArray()`.

Syntax

```
extern HRESULT sapiAddStrings  (char** StrArray,
                                ...);
```

Arguments

`strArray`	A pointer to an array of strings previously created with `sapiMakeStrArray()`. **Note:** See Memory Management for information about creating, releasing, and adding to an array of web object (URL) strings.
`...`	An ellipsis (`...`) representing a NULL-terminated variable argument list.

Returns

`HRESULT`	Each C API function returns a long integer, `HRESULT`, for the error code. Zero (`0`) indicates success; a negative integer indicates failure. Use the `SUCCEEDED` and `FAILED` macros to test for success and failure.

See Memory Management for related functions, argument descriptions, and usage information.

`sapiAddString()` adds to the specified array the strings you specify in place of the ellipsis (`...`), which denote a NULL-terminated variable argument list.

If you use this function to add strings to the `inputOptions` array, always add a name, value pair for each command option and flag, keeping in mind that each flag's value is always represented by the empty string (`""`) because flags do not have values.

Equivalent DesignSync Command

`sapiAddString()` is a C API memory management "convenience" function that does not map to any DesignSync command.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following source code example checks out the "Latest" version of all project files in a single project directory and readies them for editing, then adds three command options `[co -recursive -comment "These files are in project subdirectories" -force]` with `sapiAddStrings()` and checks out all the project files that reside in all the subdirectories beneath the project directory:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiCheckout (void* extra,
                             char** webObjectList,
                             char** inputOptions,
                             char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);
extern HRESULT sapiAddStrings (char** StrArray,
                               ...);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **objectList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and use the
```

```
// FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// checkout project files:
// [co -version "Latest" -lock]
//
// First, call MakeStrArray()to create its ObjectList array.
// The pattern that specifies the project's file naming
convention
// is *.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&objectList,
                          "*",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Call MakeStrArray() to create the inputOptions array:
// [co -version "Latest" -lock].
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          VERSION, "Latest",
                          LOCK, "",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Use sapiCheckout() to checkout the files whose URLs are in
the ObjectList
// array, applying the options/flags in the inputOptions array
and collecting
// the command results in resArray.
//
// (optional) Test for failure.
*/
```

```
    hr = sapiCheckout(NULL, objectList, inputOptions,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiCheckout failed\n");
        return E_FAIL;
    }
/*
// Call sapiAddStrArray() to add three command options to the
existing
// inputOptions array:
//( it already contains [-version "Latest" -lock]).
//
// (optional) Test for failure.
*/
    hr = sapiAddStrArray (&inputOptions,
                          RECURSIVE, "",
                          COMMENT, "These files are in project
subdirectories",
                          FORCE, "",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Use sapiCheckout() to checkout the files whose URLs are in
the
// ObjectList array, applying the options/flags in the new,
// improved, five-option inputOptions array and collecting the
command
// results in resArray.
//
// (optional) Test for failure.
*/
    hr = sapiCheckout(NULL, objectList, inputOptions,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiCheckout failed\n");
        return E_FAIL;
    }
/*
// Free the inputOptions array and the resArray (results array)
created with
// sapiMakeStrArray().
//
//(optional) Test for failure.
```

```
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiCancel()

This C API function reverses a previous checkout operation on specified locked objects.

Syntax

```
 extern HRESULT sapiCancel    (void* extra,
                               char** webObjectList,
                               char** inputOptions,
                               char*** results);
```

See the C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [cancel]

See the description of the cancel command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

This example checks out the "Latest" project files (except text files) from a single project directory, adds a comment (`"Reviewed by DGB"`) to all checked out files, and

readies them for editing. It then cancels the checkout of all files ending in `.v`, except those whose filenames begin with `new`, leaving links to files in the cache. The DesignSync cancel command options are: `[cancel -share -exclude new* *.v]`

The following source code example checks out the "Latest" version of all project files from a hierarchical set of project subdirectories. If any files of the same name already exist in the local workspace they are overwritten without exception:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiCheckout (void* extra,
                             char** webObjectList,
                             char** inputOptions,
                             char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern HHHRESULT sapiCancel (void* extra,
                             char** webObjectList,
                             char** inputOptions,
                             char*** results);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **objectList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and use the
```

```
// FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// checkout project files:
// [co -version "Latest" -exclude *.txt -comment "Reviewed by
DGB" -lock]
//
// First, call MakeStrArray()to create its ObjectList array.
// The pattern that specifies the project's file naming
convention
// is *.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&objectList,
                          "*",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Call MakeStrArray()to create the inputOptions array:
// [co -version "Latest" -exclude *.sh -recursive
//  -comment "Reviewed by DGB" -lock -force].
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          VERSION, "Latest",
                          EXCLUDE, "*.txt",
                          COMMENT, "Reviewed by DGB",
                          LOCK, "",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Use sapiCheckout() to checkout the files whose URLs are in
the
// ObjectList array, applying the options/flags in the
inputOptions
```

```
// array and collecting the command results in resArray.
//
// (optional) Test for failure.
*/
    hr = sapiCheckout(NULL, objectList, inputOptions,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiCheckout failed\n");
        return E_FAIL;
    }
/*
// Free the inputOptions array and the resArray (results array)
// created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// The cancel process code begins here. It is very similar
//to the preceding checkout code.
/*
// cancel the checkout of specified project files:
// [cancel -share -exclude new* *.v]
//
// First, call MakeStrArray()to create its ObjectList array.
// The pattern that specifies the project file naming convention
// is *.v.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&objectList,
                          "*.v",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
```

```
    }
/*
// Call MakeStrArray()to create the inputOptions array:
// [cancel -share -exclude new* *.v].
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          SHARE, "",
                          EXCLUDE, "*new*",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Use sapiCancel() to cancel the checkout of the files whose
URLs are
// in the ObjectList array, applying the options/flags in the
// inputOptions array and collecting the command results in
resArray.
//
// (optional) Test for failure.
*/
    hr = sapiCancel(NULL, objectList, inputOptions, &resArray);

    if( FAILED(hr) ) {
        printf("sapiCheckout failed\n");
        return E_FAIL;
    }
/*
// Free the inputOptions array and the resArray (results array)
// created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
```

```
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiCheckin()

This C API function performs a check-in operation on the specified objects.

Syntax

```
 extern HRESULT sapiCheckin  (void* extra,
                              char** webObjectList,
                              char** inputOptions,
                              char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [ci]

See the description of the ci command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following source code example checks out several project files from a hierarchical set of subdirectories, adds a tag to the original files, and checks the new unaltered version back in untagged (but with a new version number), retaining a local unlocked copy of each file:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
```

```
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiCheckin (void* extra,
                            char** webObjectList,
                            char** inputOptions,
                            char*** results);

extern HRESULT sapiCheckout (void* extra,
                             char** webObjectList,
                             char** inputOptions,
                             char*** results);

extern HRESULT sapiTag (void* extra,
                        char** fileList,
                        char* tag,
                        char** inputOptions,
                        char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **objectList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and use the
// FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// checkout "Latest" project files from all project
subdirectories:
// [co -version "Latest" -exclude *.sh -recursive]
//
```

```
// First, call MakeStrArray()to create its ObjectList array.
// The pattern that specifies the project's file naming
convention
// is Project_*.vbh.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&objectList,
                          "Project_*.vbh",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Call MakeStrArray()to create the inputOptions array:
// [-version "Latest" -recursive].
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          VERSION, "Latest",
                          RECURSIVE, "",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Use sapiCheckout() to checkout the files whose URLs are in
the
// ObjectList, applying all of the options/flags in the
inputOptions
// array and collecting the command results in resArray.
// (optional) Test for failure.
*/
    hr = sapiCheckout(NULL, objectList, inputOptions,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiCheckout failed\n");
        return E_FAIL;
    }
/*
// Free the inputOptions array and the resArray (results array)
// created with sapiMakeStrArray().
//(optional) Test for failure.
```

```
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Tag current (un-checkedout) version of project files in all
project
// subdirectories.
// [tag "See next version for March updates" -recursive]
// Note:
// If you tag a locked file, you are tagging the Original
version.
//
// As before, call MakeStrArray()to create objectList array
// and inputOptions array.
// (optional) Test for failure with the FAILED macro.
*/
    hr = sapiMakeStrArray(&objectList, "*", NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiTag(NULL, objectList, RECURSIVE, "",
                                   NULL,
                                   &resArray);
    if( FAILED(hr) ) {
        printf("sapiTag failed\n"); return E_FAIL;
    }

/*
// Free the inputOptions array and the results array you created
// with sapiMakeStrArray().
//
// (optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);
```

```
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

/*
// check project files back into project subdirectories and
// retain a local unlocked copy:
// [ci -recursive -keep]
// Note:
// "ci -keep" is equivalent to "co -get." Both options fetch a
local
// unlocked copy. Use this option when you don't plan to modify
the file.
//
// First, call MakeStrArray()to create the function's ObjectList
array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&objectList,
                          "Project_*.vbh",
                          NULL);
  if( FAILED(hr) ) {
        printf("Second sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Call MakeStrArray()to create the inputOptions array:
// [-recursive -keep].
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          RECURSIVE, "",
                          KEEP,"",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Use sapiCheckin() to check in the files whose URLs are in the
```

```
// ObjectList, applying all of the options/flags in the
inputOptions
// array and collecting the command results in resArray.
// (optional) Test for failure.
*/
    hr = sapiCheckin(NULL, objectList, inputOptions, &resArray);

    if( FAILED(hr) ) {
        printf("sapiCheckin failed\n");
        return E_FAIL;
    }
/*
// Free the inputOptions array and the resArray created with
// sapiMakeStrArray().
//
// (optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiCheckout()

This C API function performs a checkout operation on specified objects.

Syntax

```
 extern HRESULT sapiCheckout     (void* extra,
                                  char** webObjectList,
                                  char** inputOptions,
```

```
                                    char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [co]

See the description of the co command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following source code example checks out the "Latest" version of all project files (except text files) from a hierarchical set of project subdirectories, adds a comment ("`Reviewed by DGB`") to all checked out files, and readies them for editing. If any files of the same name already exist in the local workspace they are overwritten without exception:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiCheckout (void* extra,
                             char** webObjectList,
                             char** inputOptions,
                             char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
```

```
    HRESULT hr;
    char **resArray = NULL;
    char **webObjectList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and use the
// FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// checkout project files:
// [co -version "Latest" -exclude *.sh -recursive
//  -comment "Reviewed by DGB" -lock -force]
//
// First, call MakeStrArray()to create its ObjectList array.
// The pattern that specifies the project's file naming
convention
// is *.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObjectList,
                          "*",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Call MakeStrArray()to create the inputOptions array:
// [co -version "Latest" -exclude *.sh -recursive
//  -comment "Reviewed by DGB" -lock -force].
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          VERSION, "Latest",
                          EXCLUDE, "*.sh",
                          RECURSIVE, "",
                          COMMENT, "Reviewed by DGB",
                          LOCK, "",
                          FORCE, "",
```

```
                            NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Use sapiCheckout() to checkout the files whose URLs are in
the
// ObjectList array, applying the options/flags in the
inputOptions
// array and collecting the command results in resArray.
//
// (optional) Test for failure.
*/
    hr = sapiCheckout(NULL, objectList, inputOptions,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiCheckout failed\n");
        return E_FAIL;
    }
/*
// Free the inputOptions array and the resArray (results array)
// created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiExecuteCmd()

This C API function executes a specified Tcl command.

Syntax

```
extern HRESULT sapiExecuteCmd     (char* tclCmd,
                                   char** results);
```

Arguments

`tclCmd`	A pointer to the specified tcl command.
`results`	A pointer to a pointer to the result of the tcl command executed by this function. The topic About C API Arguments has more information.

Returns

`HRESULT`	Each C API function returns a long integer, `HRESULT`, for the error code. Zero (`0`) indicates success; a negative integer indicates failure. Use the `SUCCEEDED` and `FAILED` macros to test for success and failure.

Description

This function provides a mechanism for executing any tcl command from within a C API application. You specify a `char*` pointer to the required tcl command and a standard `char***` results pointer. The `sapiExecuteCmd()` function executes the specified tcl command and returns an `HRESULT`.

Equivalent DesignSync Command

This function does not map directly to any DesignSync command.

Sample Source Code

The following source code example executes, with `sapiExecuteCmd()`, a tcl command passed into the application as a command-line positional variable specified when the application was invoked. The application then checks the number of command-line variables with which the C API application has been invoked. If the number exceeds three (the name of the C API application itself is `argv[0]` plus two additional positional variables), an appropriate message is displayed:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiExecuteCmd (char* tclCmd,
                               char** result);

extern HRESULT sapiFreeString (char* resultString);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
     HRESULT hr;
     char *ecresult = NULL;
     . . .

/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and use the
// FAILED macro to test for failure.
*/
     hr = sapiInitialize(argc, argv);
     if( FAILED(hr) ) {
           printf("sapiInitialize: Could not initialize\n");
           return 1;
     }
/*
// Execute a tcl command that was entered on the command line
// when the C API application was invoked. Test the value of the
// argc variable.  If it is greater than two, display the
string,
// pointed to by the *ecresults pointer, then free it with
// sapiFreeString().
*/
        hr = sapiExecuteCmd(argv[2], &ecresult);
         if(argc > 2) {
             printf("Your message here: %s\n", ecresult);
```

```
        }
        sapiFreeString(ecresult);
/*
// The remainder of the C API application . . .
*/

   . . .

/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;

}
```

sapiFreeStrArray()

This C API function deallocates memory previously allocated with the `sapiAddStrArray()` function.

Syntax

```
extern HRESULT sapiFreeStrArray     (char** strArray);
```

Argument

`strArray`	A pointer to the specified string array whose memory must be deallocated.

Results

`HRESULT`	Each C API function returns a long integer, `HRESULT`, for the error code. Zero (`0`) indicates success; a negative integer indicates failure. Use the `SUCCEEDED` and `FAILED` macros to test for success and failure.

See Memory Management for related functions, argument descriptions, and usage information.

Equivalent DesignSync Command

`sapiFreeStrArray()` is a C API memory management convenience function that does not map to any DesignSync command.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following source code example checks out the "Latest" version of all project files and readies them for editing, using `sapiMakeStrArray()` and `sapiFreeStrArray()` memory management convenience functions:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiCheckout (void* extra,
                             char** webObjectList,
                             char** inputOptions,
                            char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **objectList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
```

```
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// checkout project files:
// [co -version "Latest" -exclude *.sh -recursive
//  -comment "Reviewed by DGB" -lock -force]
//
// First, call MakeStrArray()to create its ObjectList array.
// The pattern that specifies the project's file naming
// convention is *.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&objectList,
                          "*",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Call MakeStrArray() to create the inputOptions array:
// [co -version "Latest" -lock -force].
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          VERSION, "Latest",
                          LOCK, "",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Use sapiCheckout() to checkout the files whose URLs are
// in the ObjectList array, applying the options/flags in
// the inputOptions array and collecting the command results
// in resArray.
//
// (optional) Test for failure.
*/
    hr = sapiCheckout(NULL, objectList, inputOptions,
&resArray);
```

```
    if( FAILED(hr) ) {
        printf("sapiCheckout failed\n");
        return E_FAIL;
    }
/*
// Free the inputOptions array and the resArray
// (results array) created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiFreeString()

This C API function frees the specified result string.

Syntax

```
extern HRESULT sapiFreeString   (char* ResultString);
```

Argument

`ResultString`	A pointer to the specified string whose memory must be deallocated.

Returns

`HRESULT`	Each C API function returns a long integer, `HRESULT`, for the error code. Zero (`0`) indicates success; a negative integer

indicates failure. Use the `SUCCEEDED` and `FAILED` macros to test for success and failure.

See Memory Management for related functions, argument descriptions, and usage information.

Equivalent DesignSync Command

`sapiFreeString()` is a C API memory management "convenience" function that does not map to any DesignSync command.

Sample Source Code

The following source code example checks the number of command-line variables with which the C API application has been invoked. If the number exceeds three (the name of the C API application itself plus two additional positional variables), a message is displayed:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{
/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiExecuteCmd (char* tclCmd,
                               char** result);

extern HRESULT sapiFreeString (char* resultString);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
     HRESULT hr;
     char *ecresult = NULL;
     . . .

/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
```

```
*/
     hr = sapiInitialize(argc, argv);
     if( FAILED(hr) ) {
           printf("sapiInitialize: Could not initialize\n");
           return 1;
     }
/*
// Test the value of the argc variable. If greater than two,
// display the ecresult string, then free it.
*/
         hr = sapiExecuteCmd(argv[2], &ecresult);
        if(argc > 2) {
            printf("Your message here: %s\n", ecresult);
        }
        sapiFreeString(ecresult);
/*
// The remainder of the C API application . . .
*/

   . . .

/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiGetTclInterp()

This C API function returns a handle to the Tcl interpreter that is loaded with all DesignSync and Tcl commands. You can use this function to access the DesignSync and Tcl commands from a third-party tool.

Syntax

```
extern HRESULT sapiGetTclInterp  (Tcl_Interp** interp);
```

Argument

`interp`	A pointer to the returned Tcl interpreter that will be used for the DesignSync session.

Returns

HRESULT	Each C API function returns a long integer, HRESULT, for the error code. Zero (0) indicates success; a negative integer indicates failure. Use the SUCCEEDED and FAILED macros to test for success and failure.

This C API function returns a pointer to the Tcl interpreter used for the DesignSync session. The Tcl interpreter contains all DesignSync and Tcl commands, including any autoloader DesignSync Tcl commands.

Equivalent DesignSync Command

sapiGetTclInterp() is a C API memory management "convenience" function that does not map to any DesignSync command.

sapiInitialize()

This C API function initializes a DesignSync session and is required prior to calling any other C API function.

At the beginning of each C API program, you must initialize a SOM context by calling sapiInitialize(). This function also sets up the Tcl interpreter, autoloads Tcl procedures, initializes the registry files, and does the setup necessary for a DesignSync session. If you do not call this function, C API functions do not work.

Syntax

```
extern HRESULT sapiInitialize     (int argc,
                                   char** argv);
```

Arguments

argc	An integer whose value represents the total number of arguments with which the C API application was invoked.
argv	A pointer to an array of character strings that contain the arguments with which the C API application was invoked, one argument per string.

Returns

HRESULT	Each C API function returns a long integer, HRESULT, for the error code. Zero (0) indicates success; a negative integer indicates failure. The SUCCEEDED and FAILED macros defined in the scapi.h header file can be used to test for success and failure. This information is logged in the logfile if one has been

specified. Failure of a C API function can also be displayed on screen, as the following example indicates:

```
HRESULT hr;
hr = sapiInitialize(argc, argv);
if( FAILED(hr) ) {
    printf("sapiInitialize: Could not
initialize\n");
    return 1;
}
```

See Memory Management for related functions, argument descriptions, and usage information.

Equivalent DesignSync Command

`sapiInitialize()` is a C API convenience function that does not map to any DesignSync command.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

sapiLog()

This C API function controls the logging of DesignSync commands and command output.

Syntax

```
extern HRESULT sapiLog    (void* extra,
                           char* logFileName,
                           char** inputOptions,
                           char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [log]

See the description of the log command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code> presents a complete, fully commented C API sample application that calls the `sapiLog()` function.

sapiMakeBranch()

This C API function creates a new branch for the specified objects and tags each branch with the specified tag.

Syntax

```
extern HRESULT sapiMakeBranch    (void* extra,
                                  char* branchTag,
                                  char* webObject,
                                  char** inputOptions,
                                  char*** results);
```

Arguments

`extra`	Reserved for future use. For now, specify a NULL, which the `scapi.h` header file defines as a placeholder.
`branchTag`	A `char*` pointer to the tag that this function adds to each version in the newly-created branch.
`webObject`	A `char*` pointer to the string that addresses the web object (in other words, its URL).

`inputOptions`	A NULL-terminated array of name, value strings, each specifying a single command option or flag. Each name begins with a hyphen (`-`) and each array of strings is terminated by a single NULL entry.
	
	Flags are represented as name, value strings. The value is represented by an empty string (a pair of double quotes, `""`) because flags do not have an associated value.
	
	EXAMPLE:
	
	`char** inputOptions = {`
	`-option1", "value for option1 switch",`
	`"-flagX", "",`
	`"-flagY", "",`
	`NULL`
	`}`
	
	Note: See Memory Management for information about creating, releasing, and adding to an array of name, value strings.

`results`	Pointer to array of pointers to the results. The topic About C API Arguments has more information.

Returns

HRESULT	Each C API function returns a long integer, HRESULT, for the error code. Zero (0) indicates success; a negative integer indicates failure. Use the SUCCEEDED and FAILED macros to test for success and failure.

Equivalent DesignSync Command [mkbranch]

See the description of the mkbranch command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The following source code makes a branch and applies a BranchTag to the versions in the branch:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiMkBranch (void* extra,
                             char* branchTag,
                             char* webObject,
                             char** inputOptions,
                             char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char *branchTag = NULL;
```

```
    char *webObject = NULL;
    char **resArray = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Make a branch at somCheckin.txt and add a tag ("BranchTag")
// to all versions in the branch:
//
// Call sapiMkBranch() to make a branch at "branchObject,"
// applying the BranchTag and the options/flags in the
// inputOptions array and collecting the command
// results in resArray.
//
// (optional) Test for failure.
*/
    hr = sapiMkBranch (NULL,
                       "branchTag",
                       "somCheckin.txt",
                       NULL,
                       &resArray);

    if( FAILED(hr) ) {
        printf("sapiMkBranch failed\n");
        return E_FAIL;
    }
/*
// Free the resArray (results array) created with
// sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
```

```
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiMakeStrArray()

This C API function creates a pointer to a pointer to an array of strings and returns that pointer in the outgoing `strArray` argument.

Syntax

```
extern HRESULT sapiMakeStrArray   (char*** StrArray,
                                   ...);
```

Arguments

`strArray`	A `char***` pointer to a pointer to the array of strings created by this function. **Note:** See Memory Management for information about creating, releasing, and adding to an array of strings.
`...`	A NULL-terminated variable-length argument list of strings specified for inclusion in the array.

Returns

`HRESULT`	Each C API function returns a long integer, `HRESULT`, for the error code. Zero (`0`) indicates success; a negative integer indicates failure. Use the `SUCCEEDED` and `FAILED` macros to test for success and failure.

See Memory Management for related functions, argument descriptions, and usage information.

Given strings as arguments, this C API function creates and returns a pointer to an array of strings in the outgoing `strArray` argument. You should use this function to create arrays of command options for other C API functions, always adding a name, value pair for each command option and flag, and keeping in mind that each flags value is represented by the empty string (`""`) because flags do not have values.

The ellipsis (`...`) denotes a NULL-terminated variable argument list. In your source code, define `strArray` as a `char**` pointer to the specified array and pass a pointer to `strArray` (in other words a `char***`) to this function.

Use `sapiAddString()` to add strings to the resulting array.

See Memory Management for more information on this and other C API Memory Management functions.

Equivalent DesignSync Command

`sapiMakeStrArray()` is a C API memory management "convenience" function that does not map to any DesignSync command.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following source code example checks out the "Latest" version of all project files and readies them for editing, using `sapiMakeStrArray()` and `sapiFreeStrArray()` memory management convenience functions:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiCheckout (void* extra,
                             char** webObjectList,
                             char** inputOptions,
                             char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);
```

```
extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **objectList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// checkout project files:
// [co -version "Latest" -exclude *.sh -recursive
//  -comment "Reviewed by DGB" -lock -force]
//
// First, call MakeStrArray()to create its ObjectList array.
// The pattern that specifies the project's file-naming
// convention is *.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&objectList,
                          "*",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Call MakeStrArray() to create the inputOptions array:
// [co -version "Latest" -lock -force].
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          VERSION, "Latest",
                          LOCK, "",
                          NULL);
    if( FAILED(hr) ) {
```

```
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Use sapiCheckout() to checkout the files whose URLs
// are in the ObjectList array, applying the options/flags
// in the inputOptions array and collecting
// the command results in resArray.
//
// (optional) Test for failure.
*/
    hr = sapiCheckout(NULL, objectList, inputOptions,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiCheckout failed\n");
        return E_FAIL;
    }
/*
// Free the inputOptions array and the resArray (results array)
// created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following source code example checks out the "Latest" version of all project files (except text files) from a hierarchical set of project subdirectories, adds a comment ("Reviewed by DGB") to all checked out files, and readies them for editing. If any files of the same name already exist in the local workspace they are overwritten without exception:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiCheckout (void* extra,
                             char** webObjectList,
                             char** inputOptions,
                             char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **objectList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
```

```
              printf("sapiInitialize: Could not initialize\n");
            return 1;
        }
/*
// checkout project files:
// [co -version "Latest" -exclude *.sh -recursive
//  -comment "Reviewed by DGB" -lock -force]
//
// First, call MakeStrArray()to create its ObjectList array.
// The pattern that specifies the project's file-naming
// convention is *.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&objectList,
                          "*",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Call MakeStrArray()to create the inputOptions array:
// [co -version "Latest" -exclude *.sh -recursive
//  -comment "Reviewed by DGB" -lock -force].
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          VERSION, "Latest",
                          EXCLUDE, "*.sh",
                          RECURSIVE, "",
                          COMMENT, "Reviewed by DGB",
                          LOCK, "",
                          FORCE, "",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Use sapiCheckout() to checkout the files whose URLs are
// in the ObjectList array, applying the options/flags in
// the inputOptions array and collecting
// the command results in resArray.
//
// (optional) Test for failure.
*/
```

```
    hr = sapiCheckout(NULL, objectList, inputOptions,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiCheckout failed\n");
        return E_FAIL;
    }
/*
// Free the inputOptions array and the resArray
// (results array) created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following source code example checks out the "Latest" version of all project files (except text files) from a hierarchical set of project subdirectories, adds a comment ("Reviewed by DGB") to all checked out files, and readies them for editing. If any files of the same name already exist in the local workspace they are overwritten without exception:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiCheckout (void* extra,
                             char** webObjectList,
                             char** inputOptions,
                             char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **objectList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// checkout project files:
// [co -version "Latest" -exclude *.sh -recursive
//  -comment "Reviewed by DGB" -lock -force]
//
// First, call MakeStrArray()to create its ObjectList array.
```

```
// The pattern that specifies the project's file-naming
// convention is *.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&objectList,
                          "*",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Call MakeStrArray()to create the inputOptions array:
// [co -version "Latest" -exclude *.sh -recursive
//  -comment "Reviewed by DGB" -lock -force].
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          VERSION, "Latest",
                          EXCLUDE, "*.sh",
                          RECURSIVE, "",
                          COMMENT, "Reviewed by DGB",
                          LOCK, "",
                          FORCE, "",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Use sapiCheckout() to checkout the files whose URLs are
// in the ObjectList array, applying the options/flags in
// the inputOptions array and collecting
// the command results in resArray.
//
// (optional) Test for failure.
*/
    hr = sapiCheckout(NULL, objectList, inputOptions,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiCheckout failed\n");
        return E_FAIL;
    }
/*
// Free the inputOptions array and the resArray
```

```
// (results array) created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following source code example checks out the "Latest" version of all project files (except text files) from a hierarchical set of project subdirectories, adds a comment ("Reviewed by DGB") to all checked out files, and readies them for editing. If any files of the same name already exist in the local workspace they are overwritten without exception:

```
#include <stdio.h>
include <string.h>
include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
```

```
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiCheckout (void* extra,
                             char** webObjectList,
                             char** inputOptions,
                             char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **objectList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// checkout project files:
// [co -version "Latest" -exclude *.sh -recursive
//  -comment "Reviewed by DGB" -lock -force]
//
// First, call MakeStrArray()to create its ObjectList array.
// The pattern that specifies the project's file-naming
// convention is *.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&objectList,
                          "*",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
```

```
E_FAIL;
    }
/*
// Call MakeStrArray()to create the inputOptions array:
// [co -version "Latest" -exclude *.sh -recursive
//  -comment "Reviewed by DGB" -lock -force].
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          VERSION, "Latest",
                          EXCLUDE, "*.sh",
                          RECURSIVE, "",
                          COMMENT, "Reviewed by DGB",
                          LOCK, "",
                          FORCE, "",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Use sapiCheckout() to checkout the files whose URLs are
// in the ObjectList array, applying the options/flags in
// the inputOptions array and collecting
// the command results in resArray.
//
// (optional) Test for failure.
*/
    hr = sapiCheckout(NULL, objectList, inputOptions,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiCheckout failed\n");
        return E_FAIL;
    }
/*
// Free the inputOptions array and the resArray
// (results array) created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
```

```
    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
/ Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;

}
```

sapiMkFolder()

This C API function creates folder(s) (directories) on local file system or server.

Syntax

```
  extern HRESULT sapiMkFolder    (void* extra,
                                  char** folderList,
                                  char** inputOptions,
                                  char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [mkfolder]

See the description of the mkfolder command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The following source code creates six folders (directories), one of which (`asic 1 A`) has white space that must be preserved, using various kinds of relative and absolute path names, `file://` protocols, and `sync://` protocols

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{
```

```
/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **objectList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// make six folders:
//
// First, call MakeStrArray()to create the folderList array.
//
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&folderList,
                      "asic 1 A",
                      asicXXX,
                      ../asic2,
                      asic/decoder/synth,
                  // creates asic, decoder folders if necessary

                      file:////home/goss/Projects/asic,
                  // file: protocol
```

```
                    sync://chopin:2647/Projects/asic,
                // sync: protocol
                    NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Call sapiMkFolder() to create the folders described in
// the folderList array, collecting the command results
// in resArray.
//
// (optional) Test for failure.
*/
    hr = sapiMkFolder(NULL, folderList, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiCheckout failed\n");
        return E_FAIL;
    }
/*
// Free the resArray (results array) created
// with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiPopulate()

This C API function populates (fills) a directory hierarchy with files, links, or DesignSync references to files from a revision control vault.

Syntax

```
 extern HRESULT sapiPopulate   (void* extra,
                                char** inputOptions,
                                char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [populate]

See the description of the populate command in the ENOVIA Synchronicity Command Reference.

Source Code Example

This source code recursively populates the current directory, excluding all objects with a `.log` extension, in verbose mode. This operation is equivalent to the DesignSync command:

```
  populate -recursive -report verbose -exclude *.log.
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern HRESULT sapiPopulate (void* extra,
                             char** inputOptions,
                        char*** results);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
```

```
    HRESULT hr;
    char **resArray = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Recursively populate the current directory, excluding
// all objects with a ".log" extension, in verbose mode.
// [populate -recursive -report verbose -exclude *.log]
//
// First, call MakeStrArray()to create the inputOptions array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          RECURSIVE, "",
                          REPORT, verbose,
                          EXCLUDE, *.log,
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiPopulate failed\n"); return E_FAIL;
    }
/*
// Call sapiPopulate() to populate as described in the
// optionsList array, collecting the command results
// in resArray.
//
// (optional) Test for failure.
*/
    hr = sapiPopulate (NULL, optionsList, &resArray);

    if( FAILED(hr) ) {
        printf("sapiPopulate failed\n");
        return E_FAIL;
    }
/*
// Free the arrrays created with sapiMakeStrArray().
//
//(optional) Test for failure.
```

```
*/
    hr = sapiFreeStrArray(inputOptions)

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiRemoteStcl()

This C API function runs server-side stcl scripts from the client application.

Syntax

```
  extern HRESULT sapiRemoteStcl   (void* extra,
                                   char** inputOptions,
                                   char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [rstcl]

See the description of the rstcl command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The following source code executes a DesignSync command (`access reset`) that is available only as a server-side command. When specifying `sync://` protocol URLs within a script that runs on the server, do not specify the host and port.

Stopping and restarting a server is one way of reinitializing that server's access controls. The `access reset` server-side command is that way to achieve the same effect without requiring the server to be stopped and restarted.

This sample code assumes that you have created a DesignSync tcl script file (with a `.tcl` extension) containing the `access reset` command and placed it the appropriate DesignSync tcl script directory.

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern HRESULT sapiRemoteStcl (void* extra,
                               char** inputOptions,
                               char*** results);

extern void sapiShutdown ();

/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
```

```
       }
/*
// Run the access reset command on the server side.
//
// First, call MakeStrArray()to create the inputOptions array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          COMMAND, "access
reset",
                          DIR,
"$SYNC_DIR/custom/site/share/tcl",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiPopulate failed\n"); return E_FAIL;
    }
/*
// Call sapiRemoteStcl() to run the "access reset" command
// as specified in the optionsList array, collecting the
// command results in resArray.
//
// (optional) Test for failure.
*/
    hr = sapiRemoteStcl (NULL, optionsList, &resArray);

    if( FAILED(hr) ) {
        printf("sapiPopulate failed\n");
        return E_FAIL;
    }
/*
// Free the arrrays created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
```

```
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiRetire()

This C API function prevents a branch from participating in future "populate with latest versions" commands.

Syntax

```
 extern HRESULT sapiRetire    (void* extra,
                               char* webObject,
                               char** inputOptions,
                               char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [retire]

See the description of the retire command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

This sample source code demonstrates the retirement of the `surlVault.txt` file with the `-keep` command option:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);
```

```
extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern HRESULT sapiPopulate (void* extra,
                             char** inputOptions,
                             char*** results);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Retire the "surlVault.txt" file.
//
// First, call sapiMkStrArray() to create the objectList array
// and inputOptions array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&objectList, "surlVault.txt", NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
    hr = sapiMakeStrArray(&inputOptions,
                          KEEP, "",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Call sapiRetire(), specifying the two arrays
//
```

```
// (optional) Test for failure.
/*
    hr = sapiRetire(NULL,
                    objectList,
                    inputOptions,
                    &resArray);

    if( FAILED(hr) ) {
        printf("sapiRetire failed\n"); return E_FAIL;
    }
/*
// Free the array created with sapiMakeStrArray() and the
resArray.
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiRmfile()

This C API function removes (deletes) specified file(s) from local file system.

Syntax

```
 extern HRESULT sapiRmfile    (void* extra,
                               char** fileList,
                               char** inputOptions,
                               char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [rmfile]

See the description of the rmfile command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The following sample source code demonstrates the removal of several files at one time, including files specified with relative paths, absolute paths, and files whose file name begins with a hyphen (-). The equivalent DesignSync commands are:

```
  rmfile top.v /home/Projects/ASIC/top.v ../decoder.v

  rmfile -- -myfile.vb -yourfile.txt -thisfile.sh
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern HRESULT sapiRmFile (void* extra,
                           char** fileList
                           char** inputOptions,
                           char*** results);

extern void sapiShutdown ();

/*
// Declare required variables.
*/
```

```
    HRESULT hr;
    char **resArray = NULL;
    char **fileList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Remove (delete) the specified files.
// [rmfile -- -myfile.vb -yourfile.txt -thisfile.sh]
// [rmfile top.v /home/Projects/ASIC/top.v ../decoder.v]
//
//
// First call MakeStrArray()to create the first inputOptions
array.
// Call it again to create the first fileList array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          --, "",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiPopulate failed\n"); return E_FAIL;
    }

    hr = sapiMakeStrArray(&fileList,
                          -myfile.vb,
                          -yourfile.txt,
                          -thisfile.sh,
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiRmFile failed\n"); return E_FAIL;
    }
/*
// Call sapiRmFile() to remove the files described in the
// fileList array, using the options listed in the inputOptions
// array, collecting the command results in resArray.
//
// (optional) Test for failure.
```

```
*/
    hr = sapiRmFile (NULL, fileList, inputOptions, &resArray);

    if( FAILED(hr) ) {
        printf("sapiRmFile failed\n");
        return E_FAIL;
    }
/*
// Free the arrrays created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
    hr = sapiFreeStrArray(fileList);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Now remove these files:
// [rmfile top.v /home/Projects/ASIC/top.v ../decoder.v]
// Call MakeStrArray() again to create the second fileList
array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&fileList,
                          top.v,
                          /home/Projects/ASIC/top.v,
                          ../decoder.v,
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiPopulate failed\n"); return E_FAIL;
    }
/*
// Call sapiRmfile() to remove the files as described in the
// fileList array, collecting the command results in resArray.
//
// Specify the inputOptions array as NULL.
```

```
//
// (optional) Test for failure.
*/
    hr = sapiRmFile (NULL, fileList, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiRmFile failed\n");
        return E_FAIL;
    }
/*
// Free the arrays created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(fileList);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiRmfolder()

This C API function removes (deletes) a specified folder from a local or server file system.

Syntax

```
 extern HRESULT sapiRmfolder    (void* extra,
                                 char** folderList,
                                 char** inputOptions,
                                 char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [rmfolder]

See the description of the rmfolder command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The following sample source code demonstrates the removal of all files in a folder, beginning with the removal of files specified with relative paths, files specified with absolute paths, and files whose filename begins with a hyphen (-). Once the folder is empty, it, too is removed along with a previously emptied folder at the same level in the hierarchy. The equivalent DesignSync commands are:

```
rmfile topper1.vb /home/Projects/ASIC/topper2.vb ../topper3.vb

rmfile -- /home/Projects/ASIC/-myfile.vb /home/Projects/ASIC/-
yourfile.txt /home/Projects/ASIC/-thisfile.sh

rmfolder /home/Projects/ASIC
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern HRESULT sapiRmFile (void* extra,
                           char** fileList
                           char** inputOptions,
                           char*** results);
```

```
extern HRESULT sapiRmFolder (void* extra,
                             char** folderList
                             char** inputOptions,
                             char*** results);

extern void sapiShutdown ();

/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **folderList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Remove (delete) the specified files.
// [rmfile -- /home/Projects/ASIC/-myfile.vb
/home/Projects/ASIC/
//      -yourfile.txt /home/Projects/ASIC/-thisfile.sh]
// [rmfile top.v /home/Projects/ASIC/top.v ../decoder.v]
//
//
// First call MakeStrArray()to create the first inputOptions
array.
// Call it again to create the first fileList array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          --, "",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiPopulate failed\n"); return E_FAIL;
    }

    hr = sapiMakeStrArray(&fileList,
                          /home/Projects/ASIC/-myfile.vb,
```

```
                            /home/Projects/ASIC/-yourfile.txt,
                            /home/Projects/ASIC/-thisfile.sh,
                            NULL);
    if( FAILED(hr) ) {
        printf("sapiRmFile failed\n"); return E_FAIL;
    }
/*
// Call sapiRmFile() to remove the files described in the
// fileList array, using the options listed in the
// inputOptions array, collecting the command results
// in resArray.
//
// (optional) Test for failure.
*/
    hr = sapiRmFile (NULL, fileList, inputOptions, &resArray);

    if( FAILED(hr) ) {
        printf("sapiRmFile failed\n");
        return E_FAIL;
    }
/*
// Free the arrays created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
    hr = sapiFreeStrArray(fileList);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Now remove the files:
// [rmfile topper1.vb /home/Projects/ASIC/topper2.vb
../topper3.vb]
// Call MakeStrArray() again to create the second fileList
array.
//
```

```
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&fileList,
                          topper1.vb,
                          /home/Projects/ASIC/topper2.vb,
                          ../topper3.vb,
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Call sapiRmfile() to remove the files as described in the
// fileList array, collecting the command results in resArray.
//
// Specify the inputOptions array as NULL.
//
// (optional) Test for failure.
*/
    hr = sapiRmFile (NULL, fileList, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiRmFile failed\n");
        return E_FAIL;
    }
/*
// Free the arrays created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(fileList);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
//  Now remove the empty /home/Projects/ASIC folder and
// another previously emptied folder, /home/Projects/ASIC_OLD
//
//  Note:
//  In a real application you would test these folders to
// verify that they were actually empty.
//
```

```
// First, call sapiMakeStrArray() to create the folderList
array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&folderList,
                          /home/Projects/ASIC/,
                          /home/Projects/ASIC_OLD/,
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// Call sapiRmFile() to remove the folders specified
// in the folderList array.
//
// (optional) Test for failure.
*/
    hr = sapiRmFile (NULL, folderList, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiRmFile failed\n");
        return E_FAIL;
    }
/*
// Free the arrays created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(folderList);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;
```

```
return S_OK;
}
```

sapiRmvault()

This C API function removes (deletes) the specified vault without removing any corresponding files in the local work area.

Syntax

```
 extern HRESULT sapiRmvault    (void* extra,
                                char** vaultList,
                                char** inputOptions,
                                char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [rmvault]

See the description of the rmvault command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

This sample source code removes the vault for `top.v`, where `top.v` is in the current work area. The equivalent DesignSync command is:

```
  rmvault "top.v"
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);
```

```
extern HRESULT sapiFreeStrArray (char** strArray);

extern HRESULT sapiRmFile (void* extra,
                           char** fileList
                           char** inputOptions,
                           char*** results);

extern HRESULT sapiRmVault (void* extra,
                            char** vaultList
                            char** inputOptions,
                            char*** results);

extern void sapiShutdown ();

/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **vaultList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Create all necessary striing arrays and specify them when
// calling sapiRmVault()
//
//(optional) Test for failure.
*/
    hr = sapiMakeStrArray(&vaultList,
                          "top.v",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
    hr = sapiRmVault(NULL,
                     vaultList,
                     NULL,
                     &resArray);
```

```
    if( FAILED(hr) ) {
        printf("sapiRmVault failed\n"); // return E_FAIL;
    }
/*
// Free the arrays created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(vaultList);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
   sapiShutdown();

    return 0;

return S_OK;
}
```

sapiRmversion()

This C API function removes the specified version from the specified vault to free up disk space.

Syntax

```
 extern HRESULT sapiRmversion   (void* extra,
                                 char* versionList,
                                 char** inputOptions,
                                 char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [rmversion]

See the description of the rmversion command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

This sample source code removes version 1.2 of the `somRmversion.txt` file. The equivalent DesignSync command is:

```
  [rmversion "somRmversion.txt;1.2"]
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern HRESULT sapiRmFile (void* extra,
                           char** fileList
                           char** inputOptions,
                           char*** results);

extern HRESULT sapiRmVersion (void* extra,
                              char** versionList
                              char** inputOptions,
                              char*** results);

extern void sapiShutdown ();

/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **versionList = NULL;
    char **inputOptions = NULL;
/*
```

```
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Create necessary striing arrays and specify them
// when calling sapiRmVerion()
//
//(optional) Test for failure.
*/
    hr = sapiMakeStrArray(&objectList,
                          "somRmversion.txt;1.2",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiRmversion(NULL,
                       objectList,
                       NULL,
                       &resArray);
    if( FAILED(hr) ) {
        printf("sapiRmversion failed\n"); // return E_FAIL;
    }
/*
// Free the arrays created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(folderList);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
```

```
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiRun()

This C API function runs (executes) all dss or stcl commands in the specified file.

Syntax

```
 extern HRESULT sapiRun     (void* extra,
                             char* fileName,
                             char** inputOptions,
                             char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [run]

See the description of the run command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code sets the default log directory, which is also where DesignSync looks for scripts, to `/home/goss/Projects`, then executes all the DesignSync commands, ignoring command errors in `myscript.dss`. The equivalent DesignSync command is:

```
  [run -defaultdir /home/goss/Projects myscript.dss -ignoreerrs]
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{
```

```
/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiRun (void* extra,
                        char* webObject,
                        char** inputOptions,
                        char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiRun() to specify a default log directory (which
// also holds stcl script files) and run all commands in
// the specified script, ignoring command errors.
// [run -defaultdir /home/goss/Projects myscript.dss -
ignoreerrs]
//
// First, call MakeStrArray() to create two arrays:
// a one-string webObject array and an inputOptions array.
//
// (optional) Test for failure.
*/
```

```
    hr = sapiMakeStrArray(&webObject,
                          "myscript.dss",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }

    hr = sapiMakeStrArray(&inputOptions,
                          DEFAULTDIR, "/home/goss/Projects",
                          IGNOREERRS, "",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapRun()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiRun (NULL, webObject, inputOptions, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlTags failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array, the inputOptions array, and
// the resArray (results array).
//
// (optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);
```

```
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiSetmirror()

This C API function associates a mirror directory with a local working directory.

Syntax

```
 extern HRESULT sapiSetmirror     (void* extra,
                                   char* mirrorDir,
                                   char* localDir,
                                   char** inputOptions,
                                   char*** results);
```

See C API Function Prototype for argument descriptions and usage information..

Equivalent DesignSync Command [setmirror]

See the description of the setmirror command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

See sapiUrlMirror() for sample source code that calls `sapiSetMirror()` to set the mirror directory for a local directory, and then returns the mirror directory by calling `sapiUrlMirror()`.

sapiSetowner()

This C API function sets the ownership of a web object to the name specified as owner.

Syntax

```
 extern HRESULT sapiSetowner   (void* extra,
                                char* webObject,
```

```
                                    char* owner,
                                    char** inputOptions,
                                    char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [setowner]

See the description of the setowner command in the ENOVIA Synchronicity Command Reference.

sapiSetvault()

This C API function associates a specified local working directory with a specified revision control -storage vault directory.

Syntax

```
extern HRESULT sapiSetvault  (void* extra,
                              char* vaultDirURL,
                              char* localDir,
                              char** inputOptions,
                              char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [setvault]

See the description of the setvault command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiSetVault()` to associate a remote server vault directory with the current working directory. The vault directory (`Projects/apollo11`) is relative to the server root directory that was specified during server configuration (`sync://server.domain.com:2647/Projects/apollo11`). The equivalent DesignSync command is:

```
setvault sync://server.domain.com:2647/Projects/apollo11
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiSetVault (void* extra,
                             char* vaultDirURL,
                             char* localDir,
                             char** inputOptions,
                             char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *vaultDirURL = NULL;
    char *localDir = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiSetVault()
// [setvault sync://server.domain.com:2647/Projects/apollo11]
```

```
//
// First, call MakeStrArray() to create a one-string
// vaultDirURL array.
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                "sync://server.domain.com:2647/Projects/apollo11
",
                 NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiSetVault()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlVault (NULL, vaultDirURL, NULL, NULL,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiSetVault failed\n");
        return E_FAIL;
    }
/*
// Free the vaultDirURL array and the resArray (results array).
//
// (optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;
```

```
return S_OK;
}
```

sapiShutdown()

This C API function releases the DesignSync Object Model (SOM) session sink context at the end of a C API program.

Syntax

```
extern HRESULT sapiShutdown();
```

Description

You call this function at the end of each C API program to release the SOM session sink context created by `sapiInitialize()`.

See Step 9 in Standard Programming Tasks and Practices for usage information.

Equivalent DesignSync Command

This C API function does not map to any DesignSync command.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

sapiSwitchlocker()

This C API function changes the lock owner of a branch.

Syntax

```
extern HRESULT sapiSwitchlocker   (void* extra,
                                   char* newLockOwner,
                                   char* lockedBranchObject,
                                   char** inputOptions,
                                   char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [switchlocker]

See the description of the switchlocker command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

This sample source code calls `sapiSwitchLocker()` to change the lock owner of a specified branch The equivalent DesignSync command is:

```
  switchlocker goss top.v
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiSetvaultmirror (void* extra,
                                   char* newLockOwner,
                                   char* lockedBranchObject,
                                   char** inputOptions,
                                   char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char *newLockOwner = NULL;
    char *lockedBranchObject = NULL;
    char **resArray = NULL;
    char *vault = NULL;
    char **inputOptions = NULL;
/*
```

```
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiSwitchLocker() to change the owner of the lock
// on the branch object (top.v) to "goss."
// [switchlocker goss top.v]
//
//
// First, call MakeStrArray() to create a one-string
// newLockOwner array. A one-string mirrorDir array,
// and a inputOptions array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&newLockOwner,
                          "goss",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }

    hr = sapiMakeStrArray(&lockedBranchObject,
                          "top.v",
                          NULL);
    if( FAILED(hr) ) {
        printf("Second sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Call sapiSwitchLocker()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiSwitchLocker (NULL, newLockOwner,
lockedBranchObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiSetvaultmirror failed\n");
```

```
        return E_FAIL;
    }
/*
// Free the newLockOwner array, the lockedBranchObject,
// and the resArray (results array).
//
// (optional) Test for failure.
*/
    hr = sapiFreeStrArray(newLockOwner);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(lockedBranchObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiSyncinfo()

This C API function returns a name, value pair list (association list) containing information about DesignSync software.

Syntax

```
 extern HRESULT sapiSyncinfo     (void* extra,
                                  char** arglist,
                                  char** inputOptions,
                                  char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [syncinfo]

See the description of the syncinfo command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code returns the version number of the DesignSync software. The equivalent DesignSync command is:

```
  [syncinfo version]
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiSyncInfo (void* extra,
                             char** argList,
                             char** inputOptions,
                             char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
```

```
    char **resArray = NULL;
    char **argList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlSyncInfo() with the VERSION option to return
// the version number of the DesignSync software.
//
// First, call MakeStrArray() to create an inputOptions array.
//
// (optional) use the FAILED macro to test for failure.
*/
    hr = sapiMakeStrArray (&inputOptions,
                           VERSION,
                           NULL);

    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n");
        return E_FAIL;
    }

    hr = sapiSyncInfo (NULL, NULL, inputOptions, &resArray);

    if( FAILED(hr) ) {
        printf("sapiSyncInfo failed\n");
        return E_FAIL;
    }
/*
// Free the inputOptions array and the resArray (results array).
//
// (optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
```

```
    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiTag()

This C API function assigns a symbolic tag name to a single version of each specified file.

Syntax

```
 extern HRESULT sapiTag      (void* extra,
                              char** fileList,
                              char* tag,
                              char** inputOptions,
                              char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [tag]

See the description of the tag command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

sapiUnlock()

This C API function unlocks the specified file in the vault to permit other users to check in changes.

Syntax

```
extern HRESULT sapiUnlock    (void* extra,
                              char** webObjectList,
                              char** inputOptions,
                              char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [unlock]

See the description of the unlock command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

This sample source code calls `sapiUnlock()` to unlock all locked `.v` files in the current directory and recursively unlock all files in the `alu` directory. The equivalent DesignSync command is:

```
unlock *.v -recursive alu
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUnlock (void* extra,
                           char** webObjectList,
                           char** inputOptions,
                           char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);
```

```
extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **webObjectList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUnlock() to unlock specified files:
// [unlock *.v -recursive alu]
//
// First, call MakeStrArray()to create the webObjectList array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObjectList,
                          "*.v",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Call MakeStrArray()to create the inputOptions array:
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&inputOptions,
                          RECURSIVE, "alu",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
```

```
// Call sapiUnlock() to unlock the files whose URLs
// are in the webObjectList array, applying the RECURSIVE
// option in the inputOptions array and collecting
// the command results in resArray.
//
// (optional) Test for failure.
*/
    hr = sapiUnlock (NULL, webObjectList, inputOptions,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiUnlock failed\n");
        return E_FAIL;
    }
/*
// Free the arrays created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(webObjectsList);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlBranchid()

This C API function returns the branch number of the specified web object.

Syntax

```
extern HRESULT sapiUrlBranchid   (void* extra,
                                  char* webObject,
                                  char** inputOptions,
                                  char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url branchid]

See the description of the url branchid command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following source code example calls `sapiUrlBranchid()` to return the branch number of the following web object:

Object	Description	Version	Branch Tags	Returns
`test.mem`	local managed object	1.2.1.2	rel1.2	1.2.1 (current branch number)

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlBranchid (void* extra,
                                char* webObject,
```

```
                                    char** inputOptions,
                                    char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                    ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }

extern HRESULT sapiMakeStrArray (char*** strArray,
                                    ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **objectList = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and use the
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
```

```
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlBranchid on the web object whose branch
// number is required.
//
// First, call MakeStrArray()to create its one-string
// webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "test.asm",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlBranchid()
// [url branchid test.mem]
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlBranchid (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlBranchid failed\n");
        return E_FAIL;
    }
/*
// Free the one-string webObject array and the
// resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);
```

```
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlConfigs()

This C API function, when given a specified project vault folder or corresponding local working folder, returns a list of configurations for the project.

Syntax

```
 extern HRESULT sapiUrlConfigs    (void* extra,
                                   char* object,
                                   char** inputOptions,
                                   char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url configs]

See the description of the url configs command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following source code example returns the configurations for the Sportster project (`@Beta` and `@Gold`) by specifying the project folder on the server. The project on the folder is `sync://chopin:2647/Projects/Sportster` and the example returns:

```
sync://chopin:2647/Projects/Sportster@Beta
sync://chopin:2647/Projects/Sportster@Gold
```

The equivalent DesignSync command is:

```
  url configs sync://chopin:2647/Projects/Sportster



#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/

extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlConfigs (void* extra,
                               char* object,
                               char** inputOptions,
                               char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *object = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
```

```
extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *object = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);

       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlConfigs() on the object whose configs are
required.
// [url configs sync://chopin:2647/Projects/Sportster]:
//
// First, call MakeStrArray()to create its one-string object
array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&object,
                          "sync://chopin:2647/Projects/Sportster
",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlConfigs()
//
//
// (optional) Test for failure with FAILED macro.
```

```
*/
    hr = sapiUrlConfigs (NULL, object, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlConfigs failed\n");
        return E_FAIL;
    }
/*
// Free the one-string object array and the resArray (results
array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(object);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlContainer()

This C API function returns the URL of the web object (folder) that contains the specified web object (file).

Syntax

```
 extern HRESULT sapiUrlContainer   (void* extra,
                                    char* webObject,
                                    char** inputOptions,
                                    char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url container]

See the description of the url container command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following source code example returns the folder (`file:///home/Projects/Sportster/synth`) that contains the object `top.v`. The equivalent Design Sync command is:

```
  url container top.v
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlContainer (void* extra,
                                 char* webObject,
                                 char** inputOptions,
                                 char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
```

```
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);

       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlContainer() on the web object whose container
// object is required.
//
// First, call MakeStrArray()to create its one-string
// webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          " top.v",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlContainer()
// [url container top.v]
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlContainer (NULL, object, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlContainer failed\n");
        return E_FAIL;
    }
/*
// Free the one-string object array and the resArray (results
array).
//
// (optional) Test for failure.
```

```
*/
    hr = sapiFreeStrArray(object);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlContents()

This C API function returns a list of URLs of web objects (files) contained in the specified container web object (folder).

Syntax

```
 extern HRESULT sapiUrlContents  (void* extra,
                                  char* containerObject,
                                  char** inputOptions,
                                  char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url contents]

See the description of the url contents command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code assumes a local working directory named `/Projects/ASIC` that contains the following objects:

- `./Projects/FileA` # Three versions exist in the vault
- `./Projects/ASIC/FileB` # There is a new version in the vault
- `./Projects/ASIC/FileC` # Not under revision control
- `./Projects/ASIC/FileD` # In the vault, but not in local working directory
- `./Projects ASIC/Decoder/FileE` # There is a new version in the vault
- `./Projects/ASIC/Decoder/FileF` # Only file to have tag 'Gold' (version 1.2)

The sapiContents() function returns file:///Projects/ASIC/Decoder.The equivalent DesignSync command is:

```
  url contents -ifpopulated -version Gold /Projects/ASIC
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlContents (void* extra,
                                char* containerObject,
                                char** inputOptions,
                                char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
```

```
    char **resArray = NULL;
    char *containerObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlContents() on the web object whose
// contents are required.
// [url contents -ifpopulated -version Gold /Projects/ASIC]
//
// First, call MakeStrArray() twice: to create a one-string
// webObject array and to create a inputOptions array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "/Projects/ASIC",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }

    hr = sapiMakeStrArray(&containerObject,
                          IFPOPULATED, "i",
                          VERSION, "GOLD",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlContents()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlContents (NULL, containerObject, inputOptions,
&resArray);
```

```
    if( FAILED(hr) ) {
        printf("sapiUrlContents failed\n");
        return E_FAIL;
    }
/*
// Free the containerObject array, the inputOptions array,
// and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(containerObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlExists()

This C API function returns 1 if a specified web object physically exists; 0 if it does not.

Syntax

```
 extern HRESULT sapiUrlExists  (void* extra,
                                char* webObject,
                                char** inputOptions,
                                char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url exists]

See the description of the url exists command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlExists()` to determine whether two objects exist: one is a file under revision control (`top.gv` version 1.1), and the other object (`top.log`) is a DesignSync log that is not under revision control. The equivalent DesignSync commands are:

```
  url exists top.gv
  url exists top.log
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlExists (void* extra,
                              char* containerObject,
                              char** inputOptions,
                              char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
```

```
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlExists() on the web object whose existence
// must be verified.
//
// First, call MakeStrArray() to create a one-string
// webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "top.gv",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlExists()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlContents (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlContents failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
```

```
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Call sapiUrlExists() on the second web object whose
// existence must be verified.
//
// Call MakeStrArray() to create a one-string webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "top.log",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlExists()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlContents (NULL, webObject, NULL, &resArray);
    if( FAILED(hr) ) {
        printf("sapiUrlContents failed\n");
        return E_FAIL;
    }
/*
// Free the one-string webObject array and the resArray
// (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);
```

```
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlFetchedstate()

This C API function returns the fetched state of the specified object.

Syntax

```
 extern HRESULT sapiUrlFetchedstate    (void* extra,
                                        char* webObject,
                                        char** inputOptions,
                                        char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url fetchedstate]

See the description of the url fetchedstate command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlFetchedstate()` twice to return the fetched states for the following objects:

Object	Version	Fetched State

top.f	1.1 -> 1.2 [goss]	Lock
top.gv	1.2	Cache

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlFetchedstate (void* extra,
                                    char* containerObject,
                                    char** inputOptions,
                                    char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlFetchedstate() on the first web object
```

```
// whose fetched state is required.
//
// First, call MakeStrArray() to create a one-string
// webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "top.f",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }

/*
// call sapiUrlFetchedstate()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlFetchedstate (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlFetchedstate failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Call sapiUrlFetchedstate() on the second web object
// whose fetched state is required.
//
// Call MakeStrArray() to create a one-string webObject array.
```

```
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "top.gv",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }

/*
// call sapiUrlFetchedstate()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlFetchedstate (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlFetchedstate failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlFetchtime()

This C API function returns the time that the specified web object was checked out into your work area.

Syntax

```
extern HRESULT sapiUrlFetchtime   (void* extra,
                                   char* webObject,
                                   char** inputOptions,
                                   char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url fetchtime]

See the description of the url fetchtime command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code returns the fetched time for the `top.v` web object. Time is returned in `time_t` format, which is the number of seconds since the birth of UNIX -- January 1, 00:00:00, 1970 (GMT). The equivalent DesignSync command is:

```
url fetchtime top.v
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);
```

```
extern HRESULT sapiUrlFetchtime (void* extra,
                                 char* webObject,
                                 char** inputOptions,
                                 char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                  ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlFetchtime() on the web object whose
// fetch time is required.
// [url fetchtime top.v]
//
// First, call MakeStrArray() twice: to create a one-string
// webObject array and to create an inputOptions array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "top.v",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
```

```
// call sapiUrlFetchtime()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlFetchtime (NULL, webObject, inputOptions,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlFetchtime failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array, the inputOptions array, and the
// resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(containerObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
   }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlGetprop()

This C API function returns properties previously set with `sapiUrlSetprop()`.

Syntax

```
extern HRESULT sapiUrlGetprop   (void* extra,
                                 char* webObject,
                                 char* propName,
                                 char** inputOptions,
                                 char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url getprop]

See the description of the url getprop command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlGetprop()` specifying a property name of `propAAA`. The function returns a server-side property value (a string previously specified as "`property value string`"). This property value was first set on the `sync:///Projects/myproj/foo.v;` vault via a call to `sapiUrlSetprop()` function, whose use is also demonstrated:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlGetprop (void* extra,
                               char* webObject,
                               char* propName,
                               char** inputOptions,
                               char*** results);
```

```
extern HRESULT sapiUrlSetprop (void* extra,
                               char* webObject,
                               char* propName,
                               char* propValue,
                               char** inputOptions,
                               char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char *propName = NULL;
    char *propValue = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlSetProp() on the web object (the
// sync:///Projects/myproj/foo.v; vault) whose propName
// (propAAA) and propValue ("property value string") must be
set.
//
// First, call MakeStrArray() three times: to create a
// one-string webObject array, a one-string propName array,
// and a one-string propValue array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "sync:///Projects/myproj/foo.v;",
```

```
                            NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }

    hr = sapiMakeStrArray(&propName,
                            "propAAA",
                            NULL);
    if( FAILED(hr) ) {
        printf("Second sapiMakeStrArray failed\n"); return
E_FAIL;
    }

    hr = sapiMakeStrArray(&propValue,
                            "property value string",
                            NULL);
    if( FAILED(hr) ) {
        printf("Third sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlSetprop()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlSetprop (NULL, webObject, propName, propValue,
NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlFetchtime failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array, the propValue array, and
// the resArray (results array).
// Note:
// In this sample code you do not free the propName array
// because sapiUrlGetprop()requires it later on.
//
//(optional) Test for failure.
*/    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
```

```
    hr = sapiFreeStrArray(propValue);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Call sapiUrlGetprop() on the web object (the
// sync:///Projects/myproj/foo.v; vault) whose property
// value will be returned as a string.
//
// Call MakeStrArray()to create a one-string webObject array.
// The pointer to the a one-string propName array still points
// to that array because the array was never freed.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "sync:///Projects/myproj/foo.v;",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Now call sapiUrlGetprop()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlGetprop (NULL, webObject, propName, NULL,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlFetchtime failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array, the propName array, the
// propValue array, and the resArray (results array).
// Note:
// In this sample code you do not free the propName array
// because sapiUrlGetprop()requires it later on.
```

```
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(propName);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlInconflict()

This C API function returns 1 if a merge resulted in conflicts; 0 if not.

Syntax

```
 extern HRESULT sapiUrlInconflict  (void* extra,
                                    char* webObject,
                                    char** inputOptions,
                                    char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url inconflict]

See the description of the url inconflict command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code returns 1 if a merged file has conflicts; 0 if a merged file does not have conflicts.

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlInconflict (void* extra,
                                  char* webObject,
                                  char** inputOptions,
                                  char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
```

```
                printf("sapiInitialize: Could not initialize\n");
                return 1;
        }
/*
// Call sapiUrlInconflict() on the web object (myfile.txt)
// that is being tested for merge conflicts.
//
// Call MakeStrArray() to create a one-string webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "myfile.txt",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlInconflict()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlInconflict (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlInconflict failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
```

```
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlLeaf()

This C API function returns the "leaf" (text after last separator) of the specified URL.

Syntax

```
 extern HRESULT sapiUrlLeaf    (void* extra,
                                char* webObject,
                                char** inputOptions,
                                char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url leaf]

See the description of the url leaf command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code returns the "leaf" (text after last separator, in this case, the leaf is `ASIC`) from the following URL:
`sync://dvorak:2647/Projects/ASIC`:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{
/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);
```

```
extern HRESULT sapiUrlLeaf (void* extra,
                            char* webObject,
                            char** inputOptions,
                            char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
  HRESULT hr;
  char **resArray = NULL;
  char *webObject = NULL;
  char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
hr = sapiInitialize(argc, argv);
if( FAILED(hr) ) {
    printf("sapiInitialize: Could not initialize\n");
    return 1;
}
/*
// Call sapiUrlLeaf() on the web object
// (sync://dvorak:2647/Projects/ASIC) whose leaf is required.
//
// Call MakeStrArray() to create a one-string webObject array.
//
// (optional) Test for failure.
*/
  hr = sapiMakeStrArray(&webObject,
                        "sync://dvorak:2647/Projects/ASIC",
                         NULL);
  if( FAILED(hr) ) {
      printf("First sapiMakeStrArray failed\n");
      return E_FAIL;
  }
/*
// call sapiLeaf()
//
```

```
// (optional) Test for failure with FAILED macro.
*/
  hr = sapiUrlLeaf (NULL, webObject, NULL, &resArray);
  if( FAILED(hr) ) {
      printf("sapiUrlInconflict failed\n");
      return E_FAIL;
  }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
*/
  hr = sapiFreeStrArray(webObject);
  if( FAILED(hr) ) {
      printf("sapiFreeStrArray failed\n");
    return E_FAIL;
  }
  hr = sapiFreeStrArray(resArray);
  if( FAILED(hr) ) {
      printf("sapiFreeStrArray failed\n");
      return E_FAIL;
  }
/*
// Release SOM context and default settings of session sink
*/
  sapiShutdown();

  return 0;

return S_OK;
}
```

sapiUrlLocktime()

This C API function, when given a specified local object or branch, returns the exact time when the associated branch was locked.

Syntax

```
 extern HRESULT sapiUrlLocktime  (void* extra,
                                  char* webObject,
                                  char** inputOptions,
                                  char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url locktime]

See the description of the url locktime command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code returns the time that the specified local object was locked:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlLocktime (void* extra,
                                char* webObject,
                                char** inputOptions,
                                char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
```

```
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlLocktime() on the web object
// (file://dvorak:2647/Projects/ASIC/lockedfile.asm)
// whose lock time is required.
//
// Call MakeStrArray() to create a one-string webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                "file://dvorak:2647/Projects/ASIC/lockedfile.as
m",
                NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// call sapiLeaf()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlLocktime (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlLocktime failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);
```

```
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlMembers()

This C API function returns the list of members for the specified collection object.

Syntax

```
 extern HRESULT sapiUrlMembers   (void* extra,
                                  char* collectionObject,
                                  char** inputOptions,
                                  char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url members]

See the description of the url members command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code returns the members of the `symbol.sync.cds` (Cadence Cellview) collection object:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{
```

```
/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlMembers (void* extra,
                               char* collectionObject,
                               char** inputOptions,
                               char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *collectionObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlMembers() on the collection object
// (symbol.sync.cds) whose members list is required.
//
// First, call MakeStrArray() to create a one-string
// collectionObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&collectionObject,
                          "symbol.sync.cds",
                          NULL);
```

```
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// call sapiUrlMembers()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlMembers (NULL, collectionObject, NULL,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlLocktime failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(collectionObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlMirror()

This C API function returns the URL of the mirror directory previously associated with a local directory by `sapiSetmirror()`.

Syntax

```
 extern HRESULT sapiUrlMirror    (void* extra,
                                  char* localDir,
                                  char** inputOptions,
                                  char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url mirror]

See the description of the url mirror command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlSetmirror()` to set the mirror directory (`/users/admin/Projects/mirror/ASIC`) for a local working directory (`/home/goss/Projects/ASIC`), then calls `sapiUrlMirror()` to return the mirror for the same local directory:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlSetmirror (void* extra,
                                 char* mirrorDir,
                                 char* localDir,
                                 char** inputOptions,
                                 char*** results);

extern HRESULT sapiUrlSetmirror (void* extra,
                                 char* localDir,
                                 char** inputOptions,
                                 char*** results);
```

```
extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *collectionObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiSetMirror() on the local Directory object
// (/home/goss/Projects/ASIC) whose mirror directory
// (/users/admin/Projects/mirror/ASIC) is being set.
//
// First, call MakeStrArray() twice: to create a one-string
// mirrorDir array, and to create a one-string localDir array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&mirrorDir,
                          "/users/admin/Projects/mirror/ASIC",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiMakeStrArray(&localDir,
                          "/home/goss/Projects/ASIC",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
```

```
/*
// call sapiSetMirror()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlMembers (NULL, mirrorDir, localDir, NULL,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlSetMirror failed\n");
        return E_FAIL;
    }
/*
// Free the mirrorDir array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(mirrorDir);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Call sapiUrlMirror() on the local directory object
// (whose array has not yet been freed),
// to return its mirror directory.
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlMirror (NULL, localDir, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlUrlMirror failed\n");
        return E_FAIL;
    }
/*
// Free the localDir array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(localDir);
```

```
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlModified()

This C API function returns 1 if the specified object has been modified since it was fetched; 0 if it has not.

Syntax

```
 extern HRESULT sapiUrlModified (void* extra,
                                 char* webObject,
                                 char** inputOptions,
                                 char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url modified]

See the description of the url modified command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlModified()` on a specified object.The function returns a 1 if the object has been modified:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlModified (void* extra,
                                char* webObject,
                                char** inputOptions,
                                char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlModified() on the web object to determine
// whether it has been modified.
// [url fetchtime top.v]
//
```

```
// Call MakeStrArray() twice to create a one-string webObject
array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "myfile.asm",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlModified()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlModified (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlFetchtime failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;
```

```
return S_OK;
}
```

sapiUrlOwner()

This C API function returns the owner of the specified web object.

Syntax

```
 extern HRESULT sapiUrlOwner    (void* extra,
                                 char* webObject,
                                 char** inputOptions,
                                 char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url owner]

See the description of the url owner command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlOwner()` on a specified object.The function returns the owner of the specified object:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{
/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlOwner (void* extra,
                             char* webObject,
                             char** inputOptions,
                             char*** results);
```

```
extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlOwner() on the web object to return its owner.
//
// Call MakeStrArray() to create a one-string webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "myfile.asm",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlOwner()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlOwner (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlOwner failed\n");
```

```
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlPath()

This C API function strips the protocol and machine name from a URL, leaving the path. If a relative path is specified, this function returns an absolute path.

Syntax

```
 extern HRESULT sapiUrlPath    (void* extra,
                                char* url,
                                char** inputOptions,
                                char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url path]

See the description of the url path command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlPath()` on a specified URL. The function returns the path of the URL, dropping the "leaf" (text after the last delimiter):

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlPath (void* extra,
                            char* url,
                            char** inputOptions,
                            char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *url = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
```

```
                printf("sapiInitialize: Could not initialize\n");
                return 1;
        }
/*
// Call sapiUrlPath() on the URL to return its path.
//
// Call MakeStrArray() to create a one-string URL array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&url,
                          "/home/Project/mysubdir/myfile.asm",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlPath()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlOwner (NULL, url, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlPath failed\n");
        return E_FAIL;
    }
/*
// Free the URL array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(url);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
```

```
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlProjects()

This C API function returns list of public projects found at the root of the specified web object's server.

Syntax

```
 extern HRESULT sapiUrlProjects   (void* extra,,
                                   char* webObject,
                                   char** inputOptions,
                                   char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url projects]

See the description of the url projects command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlProjects()` on a specified `server:port` web object (`chopin:2647`). The function returns a list of the public projects found at the root of the `chopin:2647` server.

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
```

```
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlProjects (void* extra,
                                char* webObject,
                                char** inputOptions,
                                char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlProjects() on the URL to return its path.
//
// Call MakeStrArray() to create a one-string webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "chopin:2647",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
```

```
// call sapiUrlProjects()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlProjects (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlProjects failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlProperties()

This C API function returns an array of property name, value pairs for the specified web object.

Syntax

```
 extern HRESULT sapiUrlProperties      (void* extra,
                                        char* webObject,
                                        char** inputOptions,
                                        char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url properties]

See the description of the url properties command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlProjects()` on a specified `server:port` web object (`chopin:2647`). The function returns a list of the public projects found at the root of the `chopin:2647` server.

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlProjects (void* extra,
                                char* webObject,
                                char** inputOptions,
                                char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
```

```
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlProjects() on the URL to return its path.
//
// Call MakeStrArray() to create a one-string webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "chopin:2647",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlProjects()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlProjects (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlProjects failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);
```

```
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlProjects()` on a specified `server:port` web object (`chopin:2647`). The function returns a list of the public projects found at the root of the `chopin:2647` server.

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlProjects (void* extra,
                                char* webObject,
                                char** inputOptions,
                                char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
```

```
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;

/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlProjects() on the URL to return its path.
//
// Call MakeStrArray() to create a one-string webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "chopin:2647",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlProjects()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlProjects (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlProjects failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
```

```
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlProjects()` on a specified `server:port` web object (`chopin:2647`). The function returns a list of the public projects found at the root of the `chopin:2647` server.

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlProjects (void* extra,
                                char* webObject,
```

```
                                    char** inputOptions,
                                    char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                     ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlProjects() on the URL to return its path.
//
// Call MakeStrArray() to create a one-string webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "chopin:2647",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlProjects()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlProjects (NULL, webObject, NULL, &resArray);
```

```
    if( FAILED(hr) ) {
        printf("sapiUrlProjects failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlProperties()` on a specified web object (`chopin:2647`). The function returns a name, value list of the web objects properties:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{
```

```
/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlProperties (void* extra,
                                  char* webObject,
                                  char** inputOptions,
                                  char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlProperties() on the web object to
// return its properties.
//
// Call MakeStrArray() to create a one-string webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "myfile.asm",
                          NULL);
    if( FAILED(hr) ) {
```

```
        printf(sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// call sapiUrlProjects()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlProjects (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlProjects failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlRegistered()

This C API function returns 1 if the specified web object is registered (in other words, if the specified web object is under revision control). Otherwise, it returns 0.

Syntax

```
 extern HRESULT sapiUrlRegistered  (void* extra,
```

```
                                        char* webObject,
                                        char** inputOptions,
                                        char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url registered]

See the description of the url registered command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlRegistered()` on a specified web object and returns 1 if the specified object is under revision control:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlRegistered (void* extra,
                                  char* webObject,
                                  char** inputOptions,
                                  char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
```

```
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlRegistered() on the web object to return
// its revision control status.
//
// Call MakeStrArray() to create a one-string webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "myfile.asm",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// call sapiUrlProjects()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlRegistered (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlRegistered failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);
```

```
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlRelations()

This C API function returns the web objects on which the specified collection object is dependent.

Syntax

```
 extern HRESULT sapiUrlRelations  (void* extra,
                                   char* webObject,
                                   char** inputOptions,
                                   char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url relations]

See the description of the url relations command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlRelations()` on a specified collection object (the `accusim.design.sync.mda` Mentor collection object) and returns a list of the collection object's dependencies:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlRelations (void* extra,
                                 char* webObject,
                                 char** inputOptions,
                                 char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlRegistered() on the web object to return
// its revision control status.
//
// Call MakeStrArray() to create a one-string webObject array.
```

```
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "accusim.design.sync.mda",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// call sapiUrlRelations()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlRegistered (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlRegistered failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/

    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlResolvetag()

This C API function returns the version number associated with the specified tag.

Syntax

```
 extern HRESULT sapiUrlResolvetag   (void* extra,
                                     char* webObject,
                                     char** inputOptions,
                                     char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url resolvetag]

See the description of the url resolvetag command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlResolvetag()` on a specified web object (`top.v`) and specifies a tag. The function returns the version of `top.v` that is tagged "Gold." The equivalent DesignSync command is:

```
  url resolvetag -tag Gold top.v
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlResolvetag (void* extra,
                                  char* webObject,
```

```
                                    char** inputOptions,
                                    char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                   ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlResolvetag() on theweb object and specify a tag.
//
// Call MakeStrArray() twice: to create a one-string webObject
array
// and to create an inputOptions array
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "top.v",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
    hr = sapiMakeStrArray(&inputOptions,
                          TAG, "Gold",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
```

```
    }
/*
// call sapiUrlResolvetag()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlResolvetag (NULL, webObject, inputOptions,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlResolvetag failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array, the inputOptions array, and
// the resArray (results array).
//
// (optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlRetired()

This C API function returns 1 if the specified web object has been retired in the vault. Otherwise, returns 0.

Syntax

```
extern HRESULT sapiUrlRetired   (void* extra,
                                 char* webObject,
                                 char** inputOptions,
                                 char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url retired]

See the description of the url retired command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlRetired()` on a specified web object (`myfile.asm`). The function returns 1 if the specified object has been retired in the vault.

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlRetired (void* extra,
                               char* webObject,
                               char** inputOptions,
                               char*** results);
```

```
extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlResolvetag() on the web object and specify a tag.
//
// Call MakeStrArray() twice: to create a one-string webObject
array
// and to create an inputOptions array
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "myfile.asm",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }
/*
// call sapiUrlRetired()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlResolvetag (NULL, webObject, NULL, &resArray);
```

```
    if( FAILED(hr) ) {
        printf("sapiUrlResolvetag failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
// (optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlServers()

This C API function returns a list of servers specified in site and local server-list files, eliminating duplicates.

Syntax

```
 extern HRESULT sapiUrlServers    (void* extra,
                                   char** inputOptions,
                                   char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url servers]

See the description of the url servers command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlServers()`. The function returns a list of servers in site and local server-list files.

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
/ Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlServers (void* extra,
                               char** inputOptions,
                               char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*

// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
```

```
                return 1;
        }
/*
// Call sapiUrlServers().
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlServers (NULL, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlServers failed\n");
        return E_FAIL;
    }
/*
// Free the resArray (results array).
//
// (optional) Test for failure.
*/

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlSetprop()

This C API function places arbitrary properties on the specified web object. If you specify a note object, you can change existing properties. However, you cannot add new ones.

Syntax

```
 extern HRESULT sapiUrlSetprop    (void* extra,
                                   char* webObject,
                                   char* propName,
                                   char* propValue,
                                   char** inputOptions,
```

```
                                    char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url setprop]

See the description of the url setprop command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The sample source code that demonstrates the `sapiUrlGetprop()` function contains a call to `sapiUrlSetprop()` function, whose use is also demonstrated.

sapiUrlSyslock()

This C API function manipulates arbitrary advisory system locks by a logical lock name or a path name to an actual file.

Syntax

```
extern HRESULT sapiUrlSyslock    (void* extra,
                                  char* action,
                                  char* object,
                                  char** inputOptions,
                                  char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url syslock]

See the description of the url syslock command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The sample source code acquires the specified lock and specifies that the supplied string should be interpreted as a file path name and that the name should be resolved into a canonical path. The equivalent DesignSync command is:

```
url syslock -acquire /home/users/dave/sample.txt -canonize
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h
```

```
int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern HRESULT sapiUrlSyslock (void* extra,
                               char* action,
                               char* object
                               char** inputOptions,
                               char*** results);

extern void sapiShutdown ();

/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *action = NULL;
    char *object = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main() and
// use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Create the necessary string arrays and specify
// them when calling sapiUrlSyslock().
// [url syslock -acquire /home/users/dave/sample.txt -canonize]
//
// (optional) Test for failure.
*/
```

```
    hr = sapiMakeStrArray(&action,
                          "ACQUIRE",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiMakeStrArray(&object,
                          "/home/users/dave/sample.txt",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiMakeStrArray(&inputOptions,
                          "CANONIZE, "",
                          NULL);
    if( FAILED(hr) ) {
        printf("sapiMakeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiUrlSyslock(NULL,
                        action,
                        object,
                        inputOptions,
                        &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlSyslock failed\n"); // return E_FAIL;
    }
/*
// Free the arrays created with sapiMakeStrArray().
//
//(optional) Test for failure.
*/
    hr = sapiFreeStrArray(action);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(object);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
```

```
    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

     hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlTags()

This C API function returns a list of tags for the specified object.

Syntax

```
 extern HRESULT sapiUrlTags    (void* extra,
                                char* webObject,
                                char** inputOptions,
                                char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url tags]

See the description of the url tags command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlTags()` on a specified object (version 1.2 of `top.v`). The function returns a list of tags for the specified object. The equivalent DesignSync command is:

```
  url tags -version 1.2 top.v
```

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlTags (void* extra,
                            char* webObject,
                            char** inputOptions,
                            char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
```

```
                printf("sapiInitialize: Could not initialize\n");
                return 1;
        }
/*
// Call sapiUrlTags() on the specified version of the
// web object to return a list of its tag.
//
// Call MakeStrArray() to create two arrays: a one-string
// webObject array and an inputOptions array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "top.v",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }

    hr = sapiMakeStrArray(&inputOptions,
                          VERSION, "1.2",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlTags()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlOwner (NULL, webObject, inputOptions,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlTags failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array, the inputOptions array,
// and the resArray (results array).
//
// (optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);
```

```
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(inputOptions);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlVault()

This C API function returns the URL of the vault object associated with the specified web object.

Syntax

```
 extern HRESULT sapiUrlVault   (void* extra,
                                char* webObject,
                                char** inputOptions,
                                char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url vault]

See the description of the url vault command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlVault()` on a specified web object (`myfile.asm`). The function returns the URL of the vault object previously associated with the web object (for example, by a call to `sapiSetVault()`):

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlVault (void* extra,
                             char* webObject,
                             char** inputOptions,
                             char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
```

```
            return 1;
        }
/*
// Call sapiUrlVault() on the specified web object
// (myfile.asm) to return the associated vault object.
//
// Call MakeStrArray() to create a one-string webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "myfile.asm",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlVault()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlVault (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlVault failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
// (optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
```

```
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlVaultmirror()

This C API function returns the URL of the mirror directory previously associated with the specified vault using `sapiSetvaultmirror()`.

Syntax

```
 extern HRESULT sapiUrlVaultmirror   (void* extra,
                                      char* vault,
                                      char** inputOptions,
                                      char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url vaultmirror]

See the description of the url vaultmirror command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiSetvaultmirror()` to set the mirror (`/home/goss/mirror`) for a vault (`sync://chopin:2647/Projects/Sportster`), then calls `sapiUrlVaultMirror()`, which returns the URL of the previously associated mirror directory:

```
#include <stdio.h>
#include <string.h>
#include scapi.h

int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
```

```
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiSetvaultmirror (void* extra,
                                   char* vaultDirURL,
                                   char* mirrorDir
                                   char** inputOptions,
                                   char*** results);

extern HRESULT sapiUrlVaultmirror (void* extra,
                                   char* vault,
                                   char** inputOptions,
                                   char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char *vaultDirURL = NULL;
    char *mirrorDir = NULL;
    char **resArray = NULL;
    char *vault = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiSetVaultMirror() on the specified vaultDirURL
// (sync://chopin:2647/Projects/Sportster)
// specifying the vaults mirror directory (/home/goss/mirror).
//
// Call MakeStrArray() to create a one-string webObject array.
```

```
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&vaultDirURL,
                          "sync://chopin:2647/Projects/Sportster
",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }

    hr = sapiMakeStrArray(&mirrorDir,
                          "/home/goss/mirror",
                          NULL);
    if( FAILED(hr) ) {
        printf("Second sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Call sapiSetVaultmirror()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiSetVaultMirror (NULL, vaultDirURL, mirrorDir, NULL,
&resArray);

    if( FAILED(hr) ) {
        printf("sapiSetVaultmirror failed\n");
        return E_FAIL;
    }
/*
// Free the vaultDirURL array, the mirrorDir, and
// the resArray (results array).
//
// (optional) Test for failure.
*/
    hr = sapiFreeStrArray(vaultDirURL);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(mirrorDir);
```

```
    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Call sapiUrlVaultMirror(), specifyijng a vault
// (sync://chopin:2647/Projects/Sportster). The function
// returns the vault's mirror directory (/home/goss/mirror).
//
// Call MakeStrArray() to create a one-string vault array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&vault,
                      "sync://chopin:2647/Projects/Sportster",
                      NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// Call sapiUrlVaultmirror()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiSetVaultMirror (NULL, vault, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlVaultmirror failed\n");
        return E_FAIL;
    }
/*
// Free the vault array and the resArray (results array).
//
// (optional) Test for failure.
*/
    hr = sapiFreeStrArray(vault);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
```

```
    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlVersionid()

This C API function returns version information about the specified web object.

Syntax

```
 extern HRESULT sapiUrlVersionid    (void* extra,
                                     char* webObject,
                                     char** inputOptions,
                                     char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url versionid]

See the description of the url versionid command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlVersionid()` on a specified object (`myfile.asm`). The function returns version information about the specified object:

```
#include <stdio.h>
#include <string.h>
#include scapi.h
```

```
int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlVersionid (void* extra,
                                 char* webObject,
                                 char** inputOptions,
                                 char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
|                                 ...);

extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlVersionid() on the specified web object
// (myfile.asm) to return version information.
//
// Call MakeStrArray() to create a one-string webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
```

```
                            "myfile.asm",
                            NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlVersionid()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlVersionid (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlVersionid failed\n");
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
// (optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

sapiUrlVersions()

This C API function returns a list of URLs of all version objects associated with the specified web object.

Syntax

```
extern HRESULT sapiUrlVersions  (void* extra,
                                 char* webObject,
                                 char** inputOptions,
                                 char*** results);
```

See C API Function Prototype for argument descriptions and usage information.

Equivalent DesignSync Command [url versions]

See the description of the url versions command in the ENOVIA Synchronicity Command Reference.

Sample Source Code

The topic Sample C API Application Source Code presents a complete, fully-commented C API sample application.

The following sample source code calls `sapiUrlVersions()` on a specified object (`myfile.asm`). The function returns a list of URLs of all version objects associated with the specified object:

```
#include <stdio.h>
#include <string.h>
#include scapi.h


int main(int argc, char *argv[])
{

/*
// Declare the C API functions used in this application.
*/
extern HRESULT sapiInitialize (int argc,
                               char** argv);

extern HRESULT sapiUrlVersions (void* extra,
                                char* webObject,
                                char** inputOptions,
                                char*** results);

extern HRESULT sapiMakeStrArray (char*** strArray,
                                 ...);
```

```
extern HRESULT sapiFreeStrArray (char** strArray);

extern void sapiShutdown ();
/*
// Declare required variables.
*/
    HRESULT hr;
    char **resArray = NULL;
    char *webObject = NULL;
    char **inputOptions = NULL;
/*
// Initialize a SOM context and create a session sink.
// Pass in the argc and argv arguments from main()
//
// (optional) use the FAILED macro to test for failure.
*/
       hr = sapiInitialize(argc, argv);
       if( FAILED(hr) ) {
              printf("sapiInitialize: Could not initialize\n");
              return 1;
       }
/*
// Call sapiUrlVersions() on the specified web object
// (myfile.asm) to return URLs of all version
// objects associated with the specified object.
//

// Call MakeStrArray() to create a one-string webObject array.
//
// (optional) Test for failure.
*/
    hr = sapiMakeStrArray(&webObject,
                          "myfile.asm",
                          NULL);
    if( FAILED(hr) ) {
        printf("First sapiMakeStrArray failed\n"); return
E_FAIL;
    }
/*
// call sapiUrlVersions()
//
// (optional) Test for failure with FAILED macro.
*/
    hr = sapiUrlVersionid (NULL, webObject, NULL, &resArray);

    if( FAILED(hr) ) {
        printf("sapiUrlVersions failed\n");
```

```
        return E_FAIL;
    }
/*
// Free the webObject array and the resArray (results array).
//
// (optional) Test for failure.
*/
    hr = sapiFreeStrArray(webObject);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }

    hr = sapiFreeStrArray(resArray);

    if( FAILED(hr) ) {
        printf("sapiFreeStrArray failed\n"); return E_FAIL;
    }
/*
// Release SOM context and default settings of session sink
*/
    sapiShutdown();

    return 0;

return S_OK;
}
```

Getting Assistance

Using Help

ENOVIA Synchronicity DesignSync Data Manager Product Documentation provides information you need to use the products effectively. The Online Help is delivered through WebHelp® , an HTML-based format.

Note:

Use SyncAdmin to change your default Web browser, as specified during DesignSync tools installation. See SyncAdmin Help for details.

When the Online Help is open, you can find information in several ways:

- Use the **Contents** tab to see the help topics organized hierarchically.
- Use the **Index** tab to access the keyword index.
- Use the **Search** tab to perform a full-text search.

Within each topic, **Show** and **Hide** buttons toggle the display of the navigation (left) pane of WebHelp, which contains the Contents, Index, and Search tabs. Hiding the navigation pane gives more screen real estate to the displayed topic. Showing the navigation pane givens you access to the Contents, Index, and Search navigation tools.

You can also use your browser navigation aids, such as the **Back** and **Forward** buttons, to navigate the help system.

Note: Use SyncAdmin to change your default web browser, as specified during DesignSync tools installation.

Related Topics

Getting a Printable Version of Help

Getting a Printable Version of Help

The *C-API Programmer's Guide* is available in book format from the ENOVIA Documentation CD or the DSDocumentationPortal_Server installation available on the 3ds support website (http://media.3ds.com/support/progdir/). The content of the book is identical to that of the help system. Use the book format when you want to print the documentation, otherwise the help format is recommended so you can take advantage of the extensive hyperlinks available in the DesignSync Help.

You must have Adobe® Acrobat® Reader™ Version 8 or later installed to view the documentation. You can download Acrobat Reader from the Adobe web site.

Related Topics

Using Help

Contacting ENOVIA

For solutions to technical problems, please use the 3ds web-based support system:

http://media.3ds.com/support/

From the 3ds support website, you can access the Knowledge Base, General Issues, Closed Issues, New Product Features and Enhancements, and Q&A's. If you are not able to solve your problem using this information, you can submit a Service Request (SR) that will be answered by an ENOVIA Synchronicity Support Engineer.

If you are not a registered user of the 3ds support site, send email to ENOVIA Customer Support requesting an account for product support:

enovia.matrixone.help@3ds.com

Related Topics

Using Help

Index

B

Branch

branch number 116

changing lock owner 106

creating 59

preventing populate 82

C

C-API 1

arguments

extra argument 2, 7

overview 9

results argument 7, 9

functions

declarations 2

initialization 13

mapped to DesignSync commands 17, 29

memory allocation 11

prototype 7, 9

library

function declarations 2

programs

creating 13

Check In

C-API function 40

Checkout

C-API function 45

undoing a checkout 35

Command

executing 49, 99

log 58

Configurations

listing 119

D

DesignSync

commands

mapped to C API functions 2, 17, 29

information 109

initializing 57

DesignSync Object Model (SOM)

releasing session sink content 106

F

Fetch State

returning 132
File
deleting 84
unlocking 112
Folders
creating 74
deleting 88
finding files 122, 125
H
Header File 2
Help
contacting ENOVIA 210
printing 209
using 209
HRESULT 7
I
Initialization Function 13
L
Locking
changing lock owner 106
system locks 190
M
Main Function 13
Memory Management 11
deallocating memory 51
Merging
conflicts 143
Mirrors
associating with local directories 102
associating with vaults 199
directories 153
O
Objects
checking out 35
fetch state 132
owner 102, 160
properties 189
retiring 185
versions 203, 205
web objects 128
P
Populate
C-API function 76
preventing branch populate 82
Projects
public 165

Properties
finding 138
setting 189
S
Server
eliminating duplicates 187
listing 187
T
Tag
assigning tags 112
branches 59
listing 193
versions 182
U
URL
mirror 199
URL leaf 146
URL path 162
vault 196
version objects 205
V
Vaults
associating directory with vault directory 103
deleting 94
URL of mirror associated with vaults 199
URL of vault associated with an object 196
Versions
deleting 96
number 182, 203