CA Gen

Encyclopedia API Reference Guide

Release 8.5

Second Edition

CA Technologies Product References

This document references the following CA Technologies products:

- CA Gen

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 5: Model Information 73

Chapter 8: Association Information 113

Chapter 9: Property Information 137

Chapter 11: Checkout Information 167

Chapter 12: User and Group Information 175

Chapter 15: Count Information 219

Chapter 1: Introduction

This guide describes how to use the Encyclopedia and Workstation Application Program Interface (API) software for CA Gen. It supports the Host Encyclopedia and the Client Server Encyclopedia (CSE) for all available API platforms and databases. It also supports local access to workstation models, Host Encyclopedia, and the CSE for all available API platforms and databases.

This section contains the following topics:

Audience

This guide is intended for programmers who are developing or maintaining applications that access data managed by CA Gen Workstations, Host Encyclopedias, and CSEs.

The Encyclopedia API is a set of C functions, include files, and libraries. You should be familiar with the C programming language and the software development tools for the platform on which you are working in order to use the Encyclopedia API. You should also be familiar with the CA Gen meta-model.

Local and Remote Access Encyclopedia API

The Encyclopedia API includes a set of read and update functions that provide access to local and remote data on the CA Gen encyclopedias residing on Windows, UNIX (HP-UX and AIX), and z/OS. This feature provides the capability to create Windows, HP-UX, AIX, and z/OS applications that work with schema, model, and administrative information from both the CSE and Host Encyclopedia using standard remote data access capabilities.

The following table lists the supported platforms:

API Program Platform	Encyclopedia Platform
z/OS Host Encyclopedia DB2	Host Encyclopedia
Windows DB2	Host Encyclopedia
Windows Oracle	Windows, HP-UX, AIX Oracle
Windows SQL Server	Windows SQL Server
AIX Oracle	Windows, HP-UX, AIX Oracle
HP-UX Oracle	Windows, HP-UX, AIX Oracle

Workstation-Based Encyclopedia API

The Workstation Encyclopedia API includes read and update functions that provide access to the Toolset local encyclopedia. These functions provide the capability to extract and update model information from models in the local encyclopedia.

You can load the Toolset encyclopedia using a checkout from either a CSE or a Host Encyclopedia or by creating and editing a new model. When the toolset encyclopedia is populated through a checkout, the information available on the Toolset local encyclopedia is limited to the information scoped in a subset or model checkout. These functions allow CA Gen customers and partners to write platform and product independent software to retrieve and update encyclopedia information.

Related Documentation

CA Gen provides documentation on existing interfaces and product architectures, as well as the development of new interfaces, and APIs.

The following table lists the documentation files:

File Name	File Contents	Description
heref.chm	Host Encyclopedia Reference	This document provides specifics to get information from and place information in the Host Encyclopedia. It also provides details on the relationships between the encyclopedia and tables. The tables include Objects, Properties, Associations, and Divisions.

File Name	File Contents	Description
csref.chm	CSE Reference	This document provides specifics to get information from and place information in the CSE. It also provides details on the relationships between the encyclopedia and tables. The tables include Objects, Properties, Associations, and Divisions.
tsauto.chm	Toolset Automation	This document details the objects that may be accessed with the Toolset Automation feature and the usage of the Plug-ins feature.
odrpta.chm	Object Decomposition Report (Hierarchical Format)	This is the CA Gen Information Model. It shows the properties and object associations that are inherited and from which objects they are inherited.

Install the API and Demonstration Programs

To install the Encyclopedia API software, follow the procedures in the following guides:

- Installing the Windows API or the UNIX API, see the *Distributed Systems Installation Guide.*
- Installing the z/OS API, see the *Host Encyclopedia and Host Construction Installation Guide.*

Note: To compile and link the API demonstration programs for z/OS, see the *Host Encyclopedia and Host Construction Installation Guide.* For instructions to compile and link the API demonstration programs for Windows and UNIX, see Using Windows and UNIX APIs (see page 41).

Function Categories

Encyclopedia API functions are in the following categories:

- Encyclopedia Functions
- Model Functions
- Model Lock and Unlock
- Object Functions

- Association Functions
- Property Functions
- Subset Functions
- Checkout Functions
- User and Group Functions
- Authorization Functions
- Schema Functions
- Count Functions

Presentation Structure

The following table describes the structure used in this guide for describing each function.

Parameter	Description
Function Call	The API C function call.
Description	A description of the function.
Summary	A summary of the function with the required inputs and outputs.
Inputs	The inputs are defined.
Outputs	The outputs are defined.
Return	Each return code is presented. For further information about the return codes, see API Return Code Definitions.
Example	A set of example code.
Related Functions	Related functions.

Encyclopedia API Demo Program

The Encyclopedia API demo program demonstrates the Encyclopedia API functions in a concise, user-friendly manner. The program exercises all functions available in the Encyclopedia API.

The demo program illustrates how to apply Encyclopedia API functions in an application. You can use the examples as code references during your application development.

Using non-GUI menus, you can select the functions for use. Enter the inputs at the input prompt, and the outputs are displayed when each function completes.

Files Included in the Demo Program

The demo program includes the following files:

- Executable file
- Complete source and header files
- Link deck or a command file

Note: For more information about creating and running the demo program, see the *Distributed Systems Installation Guide*.

Start the Demo Program on z/OS

To start the demo program on z/OS, you should first create a small CLIST file like the following:

```
PROC 0
 /*CONTROL LIST CONLIST SYMLIST
   DSN SYSTEM(<DB2 subsystem ID>)
   RUN PROGRAM(APIDEMO) PLAN(<he-plan-prefix>UP)       +
   LIBRARY('CA.MVSAPI.SAMPLOAD')
   DSN END
 EXIT CODE(123)
```

Run this CLIST to start the APIDEMO program.

Start the Demo Program on Windows or UNIX

Follow these steps:

1. Type the following at a command prompt:

   apidemo for local and remote Encyclopedia read-only API

   apidemow for Workstation API (statically linked)

   apidem2w for Workstation API (dynamically linked)

   upddemo for local and remote Encyclopedia Update API

2. Press Enter.

   The Main Menu opens. A sample menu is shown next.

```
*****   MAIN MENU   ******
1 - Encyclopedia Functions
2 - Model Functions
3 - Object Functions
4 - Subset Functions
5 - Checkout Functions
6 - User / Group Functions
7 - Authorization Functions
8 - Schema Functions
9 - Count Functions
Select   <Enter to Exit>   --->
```

Use Schema Functions

The schema functions may be used without needing to log on to an encyclopedia or select a model. To use the schema functions, type “8” and press Enter.

The Schema Functions Menu opens. A sample menu is shown next.

```
******  Schema Functions  ******
 62 - EApiFetchSchemaObjTypeInfo
 63 - EApiFetchSchemaObjTypeByMnem
 64 - EApiFetchSchemaPrpTypeInfo
 65 - EApiFetchSchemaCharPrpDflt
 66 - EApiFetchSchemaIntPrpDflt
 67 - EApiFetchSchemaPrpForObjType
 68 - EApiFetchSchemaAscTypeCodeInfo
 69 - EApiFetchSchemaAscForObjType
Select   <Enter to Exit>   --->
```

Use Other Functions

All functions except the schema functions require you to either log on to an encyclopedia (for the local and remote encyclopedia read or Update APIs) or select a model (for the workstation API). The following sections describe how to log on to the Encyclopedia and select a model for the Workstation using API functions.

Follow these steps:

1. Select option 1 from the main menu to go to the Encyclopedia Functions.

   The following menu appears.

```
******   Encyclopedia Functions   ******
1  - EApiConnectToEncy
2  - EApiDisconnectEncy
3  - EApiCommit
4  - EApiLogonUserId
5  - EApiPasswordValidate
6  - EApiFetchEncyInfo
7  - EApiFetchEncyModelIds
Select   <Enter to Exit>   --->
```

2. Use option 1 to connect to the encyclopedia. Enter the database parameters as they exist in the encyclopedia iefmd.ini file.
3. Use option 4 to enter the logon user ID.

Use Workstation API Functions

Follow these steps:

1. Select option 2 from the main menu to go to the Model Functions.

   The following menu appears. A sample menu is shown next.

```
******   Model Functions ******
8 -  EApiOpenModel
9 -  EApiFetchModelInfor
10 -  EApiFetchModelCreateInfo
11 - EApiFetchModelLastUpdateInfo
12 - EApiFetchModelParentInfo
13 - EApiFetchModelChkoutInfo
14 - EApiFetchModelByName
15 - EApiLockModel
16 - EApiUnLockModel
17 - EApiFetchAllModelObjects
18 - EApiFetchAllModelTypeObjs
Select   <Enter to Exit>   --->
```

2. Select option 7 to open a model. Enter the path to the model to be opened.

All other menus

All other menus are shown in this section.

```
******   Object Functions   ******
 19 - EApiFetchObjectInfo
 20 - EApiFetchObjectCreateInfo
 21 - EApiFetchObjectAncestryInfo
 22 - EApiFetchObjectWithAncestry
 23 - EApiFetchObjChkoutParentInfo
 24 - EApiFetchSingleCharPrp
 25 - EApiFetchSingleNumericPrp
 26 - EApiFetchCharArrayPrp
 27 - EApiFetchCardOneAsc
 28 - EApiFetchCardOneAscWithObjInfo
 29 - EApiFetchCardManyAsc
 30 - EApiFetchCardManyAscWithObjInfo
 31 - EApiFetchModelNameTypeObjs
 32 - EApiFetchModelObjListByNameType
 33 - EApiFetchPrpLastUpInfo
 34 - EApiFetchParentChkoutPrpInfo
 35 - EApiFetchOrderAscOccInfo
 36 - EApiFetchAscOccLastUpdInfo
 37 - EApiFetchParentAscOccChkoutInfo
Select   <Enter to Exit>   --->
```

```
******   Subset Functions   ******
 38 - EApiFetchSubsetsInModel
 39 - EApiFetchSubsetInfo
 40 - EApiFetchSubsetCreateInfo
 41 - EApiFetchSubsetByName
 42 - EApiFetchSubsetScopingObjs
 43 - EApiFetchSubsetScopingObjInfo
 44 - EApiFetchSubsetChkoutInfo
Select   <Enter to Exit>   --->
```

```
****** Checkout Functions ******
 45 - EApiFetchChkoutsForObj
 46 - EApiFetchChkoutInfoForChkoutObj
 47 - EApiFetchChkoutChkdOutObjs
 48 - EApiFetchChkoutSubsetModelId
 Select <Enter to Exit> ->
```

```
******   User / Group Functions   ******
 49 - EApiFetchAllUsers
 50 - EApiFetchUserInfo
 51 - EApiFetchUserCreateInfo
 52 - EApiFetchAllGroups
 53 - EApiFetchGroupInfo
 54 - EApiFetchGroupCreateInfo
 55 - EApiFetchUsersInGroup
 56 - EApiFetchGroupsUsersIn
Select   <Enter to Exit>   --->
```

```
******   Authorization Functions   ******
 57 - EApiFetchUsersGrpsModelAuth
 58 - EApiFetchModelAuthInfo
 59 - EApiFetchModelAuthCreateInfo
 60 - EApiFetchUsersGrpsSubsetAuth
 61 - EApiFetchSubsetAuthCreateInfo
Select   <Enter to Exit>   --->
```

```
******   Schema Functions   ******
 62 - EApiFetchSchemaObjTypeInfo
 63 - EApiFetchSchemaObjTypeByMnem
 64 - EApiFetchSchemaPrpTypeInfo
 65 - EApiFetchSchemaCharPrpDflt
 66 - EApiFetchSchemaIntPrpDflt
 67 - EApiFetchSchemaPrpForObjType
 68 - EApiFetchSchemaAscTypeCodeInfo
 69 - EApiFetchSchemaAscForObjType
Select   <Enter to Exit>   --->
```

```
******   Count Functions   ******
 70 - EApiCountEncyModelIds
 71 - EApiCountModelObjs
 72 - EApiCountModelTypeObjs
 73 - EApiCountModelNameTypeObjs
 74 - EApiCountSubsetScopingObjs
 75 - EApiCountSubsetsInModel
 76 - EApiCountChkoutChkdOutObjs
 77 - EApiCountChkoutsForObj
 78 - EApiCountCardManyAsc
 79 - EApiCountAllUsers
 80 - EApiCountAllGroups
 81 - EApiCountUsersInGroup
 82 - EApiCountGroupsInUser
 83 - EApiCountUsersGrpsModelAuth
 84 - EApiCountUsersGrpsSubsetAuth
 85 - EApiCountSchemaPrpForObjType
 86 - EApiCountSchemaAscForObjType
Select   <Enter to Exit>   --->
```

Execute the Model Functions Example

Follow these steps:

1. Select Model Functions option 8.

```
******   Schema Functions   ******
62 - EApiFetchSchemaObjTypeInfo
63 - EApiFetchSchemaObjTypeByMnem
64 - EApiFetchSchemaPrpTypeInfo
65 - EApiFetchSchemaCharPrpDflt
66 - EApiFetchSchemaIntPrpDflt
67 - EApiFetchSchemaPrpForObjType
68 - EApiFetchSchemaAscTypeCodeInfo
69 - EApiFetchSchemaAscForObjType
Select   <Enter to Exit>   --->
******   Model Functions   ******
8 -  EApiOpenModel
9 -  EApiFetchModelInfo
10 -  EApiFetchModelCreateInfo
11 -  EApiFetchModelLastUpdateInfo
12 -  EApiFetchModelParentInfo
13 -  EApiFetchModelChkoutInfo
14 -  EApiFetchModelByName
15 -  EApiLockModel
16 -  EApiUnlockModel
17 -  EApiFetchAllModelObjects
18 -  EApiFetchAllModelTypeObjs
Select <Enter to Exit>   --->   8
```

2. Enter inputs:

   a. Inputs - for function::EApiFetchModelInfo

   b. Model ID -> 9

3. The output is displayed. Press Enter.

```
Outputs for function::   EApiFetchModelInfo
ModelName   =   PT ABLK TEST MODEL 01
ModelType   =   D
ModelStatus   =   M
Release   =   9.2.A6
UserId   =   AXE
CodePage   =   0
LanguageCode   =   850
Function return status is EAPI_SUCCESSFUL_RC
Press <Enter> to Continue
```

Note: To use the Lock and Unlock functions in the CSE, the servers must be up and running. Lock and Unlock do not function for the remote z/OS API. These functions always return a successful return code. For the Workstation API, the model ID is always 0 (zero).

Close the API Demo

Follow these steps:

1. Press Enter to exit from any menu.
2. Press Enter to exit from the Encyclopedia API Demo.

Requirements to Execute the Update Functions

This section describes various requirements for execution of the Update function. Once a model has been locked, updates are only allowed to objects within that model.

Locks

The model must be locked before you can execute an Update API. Once a model has been locked, updates are only allowed to objects within that model.

After all updates needed for that model are completed, unlock the model.

Note: For information on locking and unlocking models, see the chapter Model Lock and Unlock (see page 87).

Protected Objects

The update functions cannot update objects that are protected. If an object is protected, the code EAPI_OBJECT_PROTECTED_RC is returned. Only shared objects are validated for protection. Updates are allowed to non-shared objects even though they may be checked out.

Commits

It is recommended to call the Commit function on a regular basis when doing a large number of updates against a model. The Commit function performs final validation of the updates and, if no problems are found, commits the changes to the database.

Note: The Commit function does not release the lock on a model.

Recommendations

Validation of input names and permitted values is done as specified in the metamodel. To avoid the possibility of updating properties with invalid data, the following procedures are recommended:

- Run model validation after running the Update API against a model in an encyclopedia.
- Run a consistency check after running the Update API against a workstation model.

Chapter 2: Using Windows and UNIX APIs

Use this information if you want to develop programs that use the CA Gen Encyclopedia or Workstation Application Program Interface (API).

The Encyclopedia and Workstation APIs provide a set of C subroutines to read schema and administration information, and to read or update a model in the CSE or Workstation. These functions are provided as static and dynamic load libraries along with additional development files.

These APIs have two functional sets:

- The read-only API provides access to encyclopedia and model information but cannot change that information. The components for the read-only API are installed when their appropriate companion product is installed. The read-only Workstation API is installed with the workstation; the read-only Encyclopedia API is installed with the Encyclopedia Server.
- The Update API provides read access to encyclopedia information, and it provides read and update access to model information. Installation of the Update API is optional. The Update API is a licensed product and should be installed only if you have purchased a license for it. The Update API also depends on the read-only API. Install the Update API only if you have installed the corresponding read-only component.

This chapter covers these APIs:

- Local Windows Oracle and MS SQL Server Encyclopedia API
- Remote Windows Oracle client to Windows or UNIX Oracle server Encyclopedia API
- Remote Windows to z/OS API
- Windows Workstation API

Demo program source code and executables are installed with the read-only and Update APIs. These programs allow you to exercise the functions available in the APIs. You can select which functions to use through menus. You are prompted for inputs for each function. The completion status, or return code, and information collected by each function displays when the function completes. You can use the demo program executables to determine whether the Encyclopedia or Workstation API program is able to reach the model data source, or to explore each API function. You can also read the source code to see how the various API functions are called. The link control files and other build articles provide examples that you can use when you build your own Encyclopedia or Workstation API programs.

Installed Code

These files are installed when you install the encyclopedia or workstation API software:

- API header files
- API runtime libraries
- API executable demonstration programs
- API demonstration program source and compile and link control files

The components of the Workstation API and the Encyclopedia API are installed in directories under the %Gen xx% directory.

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

The Workstation API components are installed in %Gen xx%\Gen\samples\APIDemo

The Encyclopedia API components are installed in various directories under %Gen xx%\CSE:

- bin - executable demonstration programs and dynamically loaded libraries
- lib - static link libraries and import libraries for the dynamically loaded libraries
- demo - demonstration source code and compile or link control files
- include - API header files

API Header Files

Programs using the API functions require these header files:

- eapidef.h - API function definition header file
- encyapi.h - API variable type definitions

These header files provide mnemonic definitions for object, property, and association type codes.

- atc.h - association type code mnemonics
- otc.h - object type code mnemonics
- ptc.h - property type code mnemonics

The same set of header files is installed for the read-only and Update APIs. These header files include definitions for the Update API even when the libraries for the Update API are not installed. If you attempt to use the update functions without installing the Update API, your programs will compile successfully, but will not link or load successfully.

Programs created with the Encyclopedia API require the developer libraries supplied with your database.

Workstation API

The workstation API provides access to model data stored in the Toolset .dat files. Programs using the workstation API can be linked statically with the API and support libraries or they can be linked to use the workstation API supplied in a DLL.

The Dynamic Link Libraries and the associated import libraries supplied with the workstation API are:

Workstation API DLL	Workstation API Import Lib	Description
eapi32w.dll	eapi32w.lib	Read-only API DLL and import library
ueapi32w.dll	ueapi32w.lib	Update API DLL and import library

Static libraries for programs using the workstation API:

Workstation API Static Libraries	Description
encyapi.lib	Read-only API library
updapiw.lib	Workstation Update API library, replaces encyapi.lib for update
eapiwks.lib	Workstation support library
eapisrv.lib	Static server functions library
eapiutl.lib	Static utility functions library

The demonstration program source files and some support files are provided as examples of how to build programs using the APIs and how to use the API in those programs.

Workstation API Demo Files	Description
apidemo.c	Read-only API demo, source code
apidemow.exe	Read-only API demo, static link access Toolset data source.
apidemow.txt	Read-only API demo link file, static link access Toolset
apidem2w.exe	Read-only API demo executable, dynamic link access Toolset
apidem2w.txt	Read-only API demo link file, dynamic link access Toolset
upddemo.c	Update API demonstration source code
upddemow.exe	Update API demo executable, static link access Toolset
upddemow.txt	Update API demo link file, static link access Toolset
upddem2w.exe	Update API demo executable, dynamic link access Toolset
upddem2w.txt	Update API demo link file, dynamic link access Toolset

Workstation API - Compile and Link

Compile and link apidemow.exe or upddemow.exe using the version of Microsoft Visual Studio specified in the *Technical Requirements* document for the release of CA Gen that you are using. Compile and link the programs in a Visual Studio command window. Open the command window in Administrator mode if you want to do the compile and link in place. Otherwise, copy the entire APIDemo directory to another location where you have authorization to add, modify, and delete files. Change the directory to that directory where you copied the APIDemo directory, and do your compile and link work there.

```
set INCLUDE=%INCLUDE%;%IEF%
cl -c -Gy -Zp1 -W4 -MD -EHsc -nologo apidemo.c.

link @apidemow.txt
```

Use apidem2w.txt to link the demo using the workstation API DLL. Compile upddemo.c and link using upddemow.txt or upddem2w.txt to build the workstation update demo programs.

Windows Encyclopedia API

The Windows Encyclopedia API provides access from a program running on a Windows platform to the models stored in a Client Server Encyclopedia on the same platform (local) or another platform (remote) or a Host Encyclopedia (remote).

Windows programs that use the Encyclopedia API can be linked using API static libraries or using a Dynamic Link Library that contains all the Encyclopedia API functions. Windows programs linked using the DLL implementation of the API are linked with the import library and must be installed with one of the following API DLLs and the other DLLs listed.

Windows Encyclopedia API DLL	Windows Encyclopedia API Import Library	Description
eapi32.dll	eapi32.lib	CSE read-only API
eapi32m.dll	eapi32m.lib	HE read-only API
ueapi32.dll	ueapi32.lib	CSE Update API
ueapi32m.dll	ueapi32m.lib	HE Update API
csedb<ver>.dll *	<none>	CSE database access support
cseti<ver>.dll *	<none>	CSE TCP/IP access
iefmbt.dll	iefmbt.lib	Multi-byte characters set.

***** - <ver> is replaced with a version number.

The statically linked API programs use some of the libraries listed below, depending on the selection of read-only or Update API, and connection to a Client Server Encyclopedia or to a Host Encyclopedia.

Windows Encyclopedia API Static Libraries	Description
encyapi.lib	CSE read-only functions
updapi.lib	CSE read-only and update functions, replaces encyapi.lib
encyapim.lib	HE read-only functions
updapim.lib	HE read-only and update functions, replaces encyapim.lib
eapimvs.lib	HE support functions
eapiclt.lib	CSE/HE client functions

Windows Encyclopedia API Static Libraries	Description
eapimds.lib	CSE/HE Message Dispatcher functions
eapinet.lib	CSE/HE network support functions
eapisrv.lib	CSE/HE server support functions
eapisqc.lib	CSE database access import library
eapiutl.lib	CSE/HE utility functions
htint.lib	CSE Communications import library
iefmbt.lib	CSE/HE multi-byte character import library

Demonstration programs for read-only access to CSE and HE are provided with the read-only API installation. Demonstration programs for read and update access to Client Server Encyclopedia and Host Encyclopedia are installed with the Update API. The following files are installed with the sample/demonstration programs.

Windows Encyclopedia API Demo Programs	Description
apidemo.c	Read-only API demo source code
apidemo.exe	Read-only API demo executable, static link access CSE
apidemo.txt	Read-only API demo link file, static link access CSE
apidemo2.exe	Read-only API demo executable, dynamic link access CSE
apidemo2.txt	Read-only API demo link file, dynamic link access CSE
apidemom.exe	Read-only API demo, static link access HE
apidemom.txt	Read-only API demo link file, static link access HE
apidem2m.exe	Read-only API demo, dynamic link access HE
apidem2m.txt	Read-only API demo link file, dynamic link access HE
upddemo.c	Update API demo source code
upddemo.exe	Update API demo executable, static link access CSE
upddemo.txt	Update API demo link file, static link access CSE
upddemo2.exe	Update API demo executable, dynamic link access CSE
upddemo2.txt	Update API demo link file, dynamic link access CSE
upddemom.exe	Update API demo executable, static link access HE
upddemom.txt	Update API demo link file, static link access HE

Windows Encyclopedia API Demo Programs	Description
upddem2m.exe	Update API demo executable, dynamic link access HE
upddem2m.txt	Update API demo link file, dynamic access HE
upddemo2.exe	Update API demo executable, dynamic link access CSE

Windows Encyclopedia API - Compile and Link

Compile and link apidemo.exe or upddemo.exe using the version of Microsoft Visual Studio specified in the *Technical Requirements* document for the release of CA Gen that you are using. Open the command window in Administrator mode if you want to do the compile and link in place. Otherwise, copy the demo directory to another location where you have authorization to add, modify, and delete files. Change the directory to that directory where you copied the demo directory, and do your compile and link work there.

```
set INCLUDE=%Genxx%CSE\include;%INCLUDE%
set LIB=%Genxx%CSE\lib;%LIB%
cl -c -Gy -Zp1 -W4 -MD -EHsc -nologo apidemo.c
link @apidemo.txt
```

Use apidemo2.txt to link the demo using the Encyclopedia API DLL. Compile upddemo.c and link using upddemo.txt or upddemo2.txt to build the workstation update demo programs. Use apidemom.txt or upddemom.txt to link apidemom.exe or upddemom.exe. You should look at these link control files to see how different combinations of libraries are used to produce Encyclopedia API programs that connect to CSE or HE and use static or dynamic link libraries.

Considerations for Remote Database Access

If you want the Encyclopedia API program to access a remote database, a database in a different machine, you must be aware of the database's communications requirements.

Remote Oracle Access

To access a remote Oracle database, install and configure the Oracle client software and the SQL+NET software according to the Oracle user documentation. Install the client software on the workstation on which the Encyclopedia API program will run. Install and configure the SQL+NET server software on the server running the CSE. Refer to Oracle documentation for instructions on verifying the connection between the client and server machines.

Remote DB2 Access

To access the Host Encyclopedia from a Windows workstation, install and configure DB2 Connect according to IBM's Universal Database documentation. DB2 Connect must be installed on:

- The client workstation
- z/OS as the server system

The z/OS DB2 database administrator must authorize a DB2 collection for binding API DBRM packages to the Host Encyclopedia tables.

Note: Verify the DB2 remote access is established using the DB2 Command Center before using the CA Gen APIs. This query lists the models in the Host Encyclopedia:

```
connect to <DB2 database subsystem> user <TSO user ID>
select model_name from <encyclopedia table owner ID>.dmdl
terminate
```

1. Open a DB Command window and enter the following commands in the DB2 CLP window:

   ```
   >cd %iefcsgen%\..\cse_mvsdb2
   >db2 connect to <DB2 subsystem ID> user <your TSO user ID>
   >db2 bind @mvsbind.lst blocking all sqlerror continue messages bind.msg owner
   <owner> qualifier <qualifier> collection <collection>
   >db2 connect reset
   ```

2. Close the DB2 CLP window when the work is done.

You will be prompted for your TSO password when the db2 connect command is entered. The entry field is non-display, so your password security is preserved. Enter your password and click the Return key to continue. The values in the angle brackets, < >, are replaced with owner, qualifier, and collection values that were established when the Host Encyclopedia was installed. You can get these values from your Host Encyclopedia Administrator or DB2 Administrator.

UNIX Encyclopedia API

The UNIX Encyclopedia API provides access from a program running on a Client Server Encyclopedia-supported UNIX platform to the models stored in a CSE on the same platform (local) or another platform (remote) with the same database type. On the UNIX platform, Oracle is the only supported remote database type.

UNIX programs that use the Encyclopedia API are linked using API static libraries that contain all of the Encyclopedia API functions. These programs depend on shared libraries that implement communication services and access to the selected database. The read-only libraries are installed with the Encyclopedia Servers, the update libraries are installed only when the Update API is installed.

UNIX Encyclopedia Libraries	Description
encyapi.a	Static read-only API library
updapi.a	Static Update API library, replaces encyapi.a in update programs
eapiclt.a	Static client functions library
eapimds.a	Static Message Dispatcher access library
eapinet.a	Static network support functions library
eapisrv.a	Static server functions library
eapiutl.a	Static utility functions library
eapisqc.a	Static Oracle database access library
sqlintf.a	Shared library common database access interface library
libcti<ver>.sl *	Shared library TCP / IP communications functions, HP-UX
libcdb<ver>.sl *	Shared library database access functions, HP-UX
libcti<ver>.a *	Shared library TCP / IP communications functions, AIX
libcdb<ver>.a *	Shared library database access functions, AIX

* <ver> is replaced by a version number

The demonstration program executables and source for the read-only API are installed with the Encyclopedia Servers. The demonstration program executables and source for the Update API are installed with the Update API component.

UNIX Encyclopedia API Demo Program	Description
apidemo	Read-only API demo executable
epidemo.c	Read-only API source code
upddemo	Update demo executable
upddemo.c	Update demo source code

UNIX Encyclopedia API Demo Program	Description
eapi.sh	Shell script builds apidemo, and upddemo if installed.

You can compile and link the demo programs by running the script file eapi.sh from a UNIX shell.

```
sh eapi.sh
```

The script file requires that the environment variable IEFCSGEN is defined and referenced the cse/bin directory. The script builds the read-only demo, and if it finds that the Update API is installed, the script builds the update demo as well. You should read this file to see the compiler options used to compile and link applications that use the Encyclopedia Read-only or Update API.

Chapter 3: API Variable Types

This section lists the API variable types and describes the characteristics of each type.

This section contains the following topics:

Enum Definitions

Data types listed within the enum definitions each have a set of declared enumerators. Each enumerator represents a numeric or a character value. Refer to the encyapi.h header file for specific enumerators and their values.

The following are the enum definition data types:

ADDMDLAUTH	Add Model Authorization
ADDUSRAUTH	Add User Authorization
ASCAGGACT	Association Aggregate Action Value
ASCCARD	Association Cardinality
ASCDIR	Association Direction
ASCMODREF	Association Modify/Referencing Flag
ASCORDER	Association Order
ASCREFRESH	Association Refresh
CHGSTATUS	Change Status
CKOSTATUS	Object Checkout Status
CKOUTSTATUS	Subset Checkout Status
EAPIRC	Encyclopedia API Return Codes
ENCYLOCKTYPE	Model Lock Type
GENERATEAUTH	Generate Authorization
LOCKEDLEVEL	Lock Level

ADDMDLAUTH	Add Model Authorization
MAXACCESS	Object Maximum Access
MIGRATEAUTH	Migrate Authorization
MODELSTATUS	Model Status
MODELTYPE	Model Type
OBJAGGBND	Object Aggregate/Boundary Flag
OBJSCOPING	Scoping Object Flag
PRPFORMAT	Property Format
SCOPEXP	Subset Scoping Object Expansion
SCOPPROT	Subset Scoping Object Protection
SUBSETTYPE	Subset Type
UPDATEAUTH	Update Authorization
USERADMIN	Encyclopedia Administrator

Numeric Definitions

Numeric definitions are integer values. They can be classified as double, long, or short. The length for each of these classifications varies depending on the machine and compiler used. The following are the numeric definition data types:

Double	
TIMESTAMP	Timestamp
Long	
ASCSEQ	Ordered Association Sequence
CKOID	Checkout ID
ENCYID	Encyclopedia ID
INTVALUE	Integer Value
LCOUNT	Long Count
LOCKID	Lock ID
MODELID	Model ID
OBJID	Object ID
SUBSETID	Subset ID

Double	
Short	
ASCTYPECODE	Association Type Code
CODEPAGE	Code Page
DIVTYPECODE	Division Type Code
LANGUAGECODE	Language Code
OBJPHYSSTR	Object Physical Structure
OBJTYPECODE	Object Type Code
PRPCOLUMN	Property Column
PRPLENGTH	Property Length
PRPTYPECODE	Property Type Code

Character Array Definitions

Character array definitions are text strings. Each type definition has an allowable number of characters for the text and a null termination.

The following is a list of the character array data types:

CHARVALUE	Character Value
DATESTRING[11]	Date String
DBPARMS[80]	Database Parameters
GRPID[9]	Group ID
MNEMONIC[9]	Mnemonic
NAME[33]	Name
PASSWORD[9]	Password
RELEASE[9]	Release
TEXT[32]	Text
TEXTVALUE[3951]	Text Value
TIMESTRING[9]	Time String
USERID[9]	User ID
USERNAME[33]	User Name

Count and Array Functions

Count and array functions are used together to retrieve information from the encyclopedia. The count function determines the quantity of items that exists in the encyclopedia while the array function returns a specified set of those items.

For every array function there is a complimentary count function. Use the count function first to identify the maximum available quantity. Then use this quantity (or less) as the input to the array function.

Array Function	Count Function
EApiFetchEncyModelIds	EApiCountEncyModelIds
EApiFetchAllModelObjects	EApiCountModelObjs
EApiFetchAllModelTypeObjs	EApiCountModelTypeObjs
EApiFetchModelObjListByNameType	EApiCountModelNameTypeObjs
EApiFetchSubsetsInModel	EApiCountSubsetsInModel
EApiFetchSubsetScopingObjs	EApiCountSubsetScopingObjs
EApiFetchChkoutsForObj	EApiCountChkoutsForObj
EApiFetchChkoutChkdOutObjs	EApiCountChkoutChkdOutObjs
EApiFetchAllUsers	EApiCountAllUsers
EApiFetchAllGroups	EApiCountAllGroups
EApiFetchUsersGrpsModelAuth	EApiCountUsersGrpsModelAuth
EApiFetchUsersGrpsSubsetAuth	EApiCountUsersGrpsSubsetAuth
EApiFetchUsersInGroup	EApiCountUsersInGroup
EApiFetchGroupsUsersIn	EApiCountGroupsInUser
EApiFetchSchemaPrpForObjType	EApiCountSchemaPrpForObjType
EApiFetchSchemaAscForObjType	EApiCountSchemaAscForObjType
EApiFetchCardManyAsc	EApiCountCardManyAsc

Each function requires a number for inputs and outputs.

Inputs

Input the number you want to retrieve.

```
number specified < number in ency
```

If the number specified is less than the number of items in the encyclopedia you are requesting, you get the following return code:

```
EAPI_MAXIMUM_COUNT_REACHED_RC
number specified ³ number in ency
```

If the number specified is greater than or equal to the number of items in the encyclopedia you are requesting, you get an output equal to the number of items in the encyclopedia. The return code follows:

```
EAPI_SUCCESSFUL_RC
```

If the maximum value is less than the total number of IDs, you cannot retrieve another group of IDs unless you specify a larger maximum value. The only way to retrieve the remaining IDs is to make another request with a larger maximum value. For example:

Number of Model IDs in the Encyclopedia	Input	Output
10	10	10
10	5	5
5	10	5

Memory Calculations for Fetches

Memory for the array must be allocated prior to using this function. You must allocate enough memory for the array to avoid writing to memory space that is not allocated. The count functions can assist in calculating the amount of storage required.

Chapter 4: Encyclopedia Information

This chapter describes the functions to connect and disconnect from the encyclopedia, commit and rollback the updates to the database, and fetch Encyclopedia information.

This section contains the following topics:

EApiConnectToEncy

Description

Connect to the Encyclopedia. This function allows connection to the encyclopedia through the database.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiConnectToEncy(	PDBPARMS	pszDBParms);

Input

PszDBParms

This information is required to connect to the Encyclopedia database. The required parameters are DBMS dependent. The database name, user ID, and password may be required, depending on the DBMS you are using. If you are using the Client Server Encyclopedia, refer to the <iefmd>.ini file's Encyclopedia Server unit parameters for the required values. Provide the input as a single string identifying each variable for your encyclopedia database.

For example:

"DBNAME=DBIEFD"

For z/OS DB2 access, you can optionally use the concept of package, or collection, for connection purposes. In this case, the string is modified to include a collection ID.

For example:

"DBNAME=DBIEFD/COLLECTION-ID"

Note: The separator "/" is mandatory, and must not be surrounded by spaces.

If collection is not used, the parameter uses the database name only.

Output

None.

Return

EAPI_SUCCESSFUL_RC
EAPI_CONNECT_FAILED_RC

Related Functions

EApiDisconnectEncy

Example

```
#include <eapidef.h>
```

For remote z/OS DB2 access:

(Without Collection)

```
EAPIRC rc = EAPI_SUCCESSFUL_RC;
DBPARMS szDBParms;
/* The following function connects to the encyclopedia */
strcpy(szDBParms, "DBNAME=DBIEFD");
rc = EApiConnectToEncy(szDBParms);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiConnectToEncy\n");
else
{
    /* The following function disconnects from the encyclopedia */
    rc = EApiDisconnectEncy();
    if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiDisconnectEncy\n");
)
```

With Collection, the DBNAME parameter in the previous example is:

```
strcpy(szDBParms, "DBNAME=DBIEFD/Collection-id");
```

Other parameters are the same as before.

For local Windows Oracle access:

```
strcpy(szDBParms, "DBNAME=DBIE, DBUSER=ency,
DBPSWD=ency");
```

For remote Oracle access from Windows:

```
strcpy(szDBParms, "DBNAME=DBIE,
DBUSER=ency@<sqlnetalias>,
DBPSWD=ency);
```

For UNIX Oracle:

```
strcpy(szDBParms, "DBNAME=DBIEFD,
DBUSER=ency@<sqlnetalias>,
DBPSWD=ency);
```

EApiDisconnectEncy

Description

Disconnect from Encyclopedia. You must disconnect from the currently connected database (i.e. encyclopedia) before you can connect to a new database. An error will occur if you try to connect to a new database before disconnecting.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiDisconnectEncy(	void);

Input

None.

Output

None.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_CONNECT_FAILED_RC
```

Related Functions

EApiConnectToEncy

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
DBPARMS szDBParms;
/* The following function connects to the encyclopedia */
strcpy(szDBParms, "DBNAME=DBIEFD");
rc = EApiConnectToEncy(szDBParms);
```

```
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiConnectToEncy\n");
else
{
    /* The following function disconnects from the encyclopedia */
    rc = EApiDisconnectEncy();
    if ( rc != EAPI_SUCCESSFUL_RC )
        printf("Error in EApiDisconnectEncy\n");

}
```

EApiCommit

Description

Commit. This function updates the database with any pending changes. This function can also be used to release any database locks.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCommit(	void);

Input

None.

Output

None.

Return

None.

Related Functions

None.

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
rc = EApiCommit();
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCommit\n");
```

EApiCommitUpdate

Description

This function commits updates to the database. Before doing the commit, it will validate the names of new objects added and the names of any objects whose name property was updated. If a duplicate name is found, the object will be renamed.

It is recommended that the commit API be called on a regular basis when doing a large number of updates against a model. The commit API will perform final validation of the updates and, if no problems are found, commit the changes to the database. The commit API does not release the lock on a model.

Change capture is used for this function.

Name validation rules are specified in the appendix Object Name Validation Rules.

Summary

EAPIRC EApiCommitUpdate(	CHARVALUE	* pVendor);

Input

CHAR[32]

The VendorString is a 32 byte string that should be passed into this function to indicate the vendor product that is making the update. If this field is left blank, it will default to ENCYAPI.

Output

None.

Return

EAPIRC returns the following information.

```
EAPI_SUCCESSFUL_RC if changes were committed
EAPI_MODEL_NOT_LOCKED_RC if the model is not locked for update
EAPI_NOT_CONNECTED_RC if not connected to encyclopedia
EAPI_USER_NOT_LOGGED_ON_RC if user is not logged on
```

Related Functions

EapiUpRollback

Example

```
#include<eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
char szName[32] = "VendorString";
rc = EApiCommitUpdate(szName);
if ( rc != EAPI_SUCCESSFUL_RC )
printf("Error in EApiCommitUpdate\n");
```

EApiUpRollback

Description

Rollback updates. This function undoes any updates before committing.

Summary

EAPIRC EApiUpRollback( );

Input

None.

Output

None.

Return

EAPIRC returns the following information.

```
EAPI_SUCCESSFUL_RC if changes were committed
EAPI_MODEL_NOT_LOCKED_RC if the model is not locked for update
EAPI_NOT_CONNECTED_RC if not connected to encyclopedia
EAPI_USER_NOT_LOGGED_ON_RC if user is not logged on
```

Related Functions

EapiCommitUpdate

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
rc = EApiUpRollback();
if ( rc != EAPI_SUCCESSFUL_RC )
printf("Error in EApiUpRollback\n");
```

EApiLogonUserId

Description

Logon user ID. The user must be logged on before encyclopedia data access is allowed. This function is used after EApiConnectToEncy and before any of the data access functions can be used. The user ID is any valid logon name for the Encyclopedia.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiLogonUserId(	PUSERID	pszUserId);

Input

pszUserId

The user ID is any valid logon name for the Encyclopedia. In this case, the user ID is the name you use to log on to the Encyclopedia.

Note: The user ID is case sensitive and must be in uppercase.

Output

None.

Return

EAPI_SUCCESSFUL_RC
EAPI_USER_NOT_FOUND_RC
EAPI_USER_NOT_AUTH_RC
EAPI_NOT_CONNECTED_RC

Related Functions

- EapiConnectToEncy
- EApiDisconnectToEncy

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
USERID szDBParms;
/* After connecting to the encyclopedia */
strcpy(szUserId, "CHRIS");
rc = EApiLogonUserId(szUserId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiLogonUserId\n");
```

EApiFetchEncyInfo

Description

Fetch Encyclopedia Information. This function retrieves the encyclopedia name and ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchEncyInfo(	PENCYID	pnEncyId,
	PNAME	pszEncyName);

Input

None.

Output

pnEncyID

The Encyclopedia ID is the integer that identifies the Encyclopedia.

pszEncyName

The Encyclopedia name is the string that identifies the Encyclopedia.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_ENCY_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

None.

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
ENCYID nEncyId,
NAME szName;
rc = EApiFetchEncyInfo(&nEncyId,
szName);
```

```
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchEncyInfo\n");
else
{
    printf("Encyclopedia Name = %s\n", szName);
    printf("Encyclopedia Id = %s\n", nEncyId);
}
```

EApiFetchEncyModelIds

Description

Fetch Encyclopedia Model IDs. This function retrieves all (or fewer) of the model IDs in the encyclopedia. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchEncyModelIds(	PLCOUNT	plCount,
	PMODELID	plArrayModelIds);

Input

plCount

The count is the number of models. Specify the maximum number of model IDs you want to retrieve. Specify negative one (-1) if you want to retrieve all the model IDs.

Output

plCount

This count is the number of model IDs retrieved. The number of model IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

plArrayModelIds

Model ID Array is a series that contains the returned model IDs. Memory for the array must be allocated prior to using this function.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Return

EAPI_SUCCESSFUL_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
EAPI_MAXIMUM_COUNT_REACHED_RC

Related Functions

EApiCountEncyModelIds

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
LCOUNT lCount, i;
MODELID nArrayModelIds[100];
lCount = 100L;

rc = EApiFetchEncyModelIds(&lCount,
nArrayModelIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchEncyModelIds\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i<lCount, i++)
    printf("Model Id[%ld] = %ld\n", i, nArrayModelIds[i]);

}
```

EApiPasswordValidate

Description

EApiPasswordValidate verifies that the given password is the password stored in the Coordination database. The given password is compared to the password stored in the coordination database for the given user ID. If the two passwords are the same, a successful indication is returned. This function is independent of database connection state and whether or not EApiLogonUserId has been called. It does require that the environment variable IEF_MDNAME is defined and that the CSE Server it identifies is running.

A valid password for a given user ID is the same password that would be used to logon to a CSE through a CSE client. Use CSE Administration tools to change the password associated with the given user ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiPasswordValidate(	PUSERID	szUserId,
	PPASSWORD	szpwd);

Input

szUserId

Represents an Encyclopedia user ID. It is typically the same value used in the CSE client logon dialogs.

Note: The value is case-sensitive and must be entered in uppercase.

szPwd

Is the same string of letters and other characters that are entered in the Password field of the CSE client logon panel.

Output

None.

Return

EAPI_SUCCESSFUL_RC
EAPI_INVALID_DIR_PASSWORD_RC

Related Functions

EApiLogonUserId

Example

```
EAPIRC rc;
USERID szUserID = "USERID";
PASSWORD szPassword = "PassWord10";
rc =EApiPasswordValidate(szUserID, szPassword);
if ( rc != API_SUCCESSFUL_RC )
    printf("Password validation failed for %s\n", szUserID);
```

How EApiPasswordValidate API Validates Password

The PasswordValidate entry point in the Encyclopedia API provides access to the password validation service for customer written applications that use the Encyclopedia API. This entry point accepts a user name and a password in plain text and returns an indication of whether the given password is the password on file for the given user. It also provides other diagnostic return codes to identify faults detected during the validation process.

This procedure does not use a direct database connection to the Coordination database. Instead, it sends a message to the Coordination database server, much like the CSE clients. The reply contains a return value that indicates whether the password is valid for the given user name.

Note: It is not possible to change the password through the Encyclopedia API.

The following tasks are performed by the EApiPasswordValidate API for the validation of the password:

Follow these steps:

1. Connect to the message dispatcher and registers with it.
2. Insert the userid and password into a packet and send the packet to the coordination server for validation.

3. The Coordination server validates the password and returns success, if the password matches the one with that exists in the database otherwise returns the corresponding error message.
4. The reply is received by the Encyclopedia API and control is returned to the calling program with the appropriate return code.

Chapter 5: Model Information

This section contains the following topics:

EApiOpenModel

Description

Open a workstation model. This function will open a workstation model when given a path to the model data files.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiOpenModel(	PDBPARMS szModelPath,
	CHAR * errfile);

Input

szModelPath

This is the path to the model directory containing the model data files (that is, %GENxx%\Gen\samples\models\sample.ief).

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

errfile

This is the path to a file for writing out fatal error messages. This file does not have to exist.

Output

None.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_MODEL_LOCKED_RC
EAPI_MODEL_INCOMPATIBLE_RC
EAPI_MODEL_FATAL_ERROR_RC
EAPI_MODEL_DIR_READ_RC
EAPI_MODEL_BAD_ERRFILE_RC
```

Example

```
EAPIRC rc = EAPI_SUCCESSFUL_RC;
char[80] model_dir;
char[80] errfile;
strcpy(model_dir, "%GENxx%\\Gen\\sample.ief”);
strcpy(errfile, "c:\\ief\\err.log”);
rc = EApiOpenModel(model_dir, errfile);
if (rc != EAPI_SUCCESSFUL_RC)
```

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

The model ID is not returned from this function. The model ID is always 0 (zero) for a free workstation model when calling other API routines that require a model ID.

EApiFetchModelInfo

Description

Fetch Model Information. This function retrieves model information from the encyclopedia. The information returned is model: name, type, status, release, owner user ID, code page and language code.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchModelInfo(	MODELID	nModelId,
	PNAME	pszModelName,
	PMODELTYPE	peType,
	PMODELSTATUS	peStatus,
	PRELEASE	pszRelease,
	PNAME	pszOwnerUserName,
	PCODEPAGE	pnModelCPage,
	PLANGUAGECODE	pnModelLangCode);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model. This designates the model from which you want to retrieve the information.

Output

pszModelName

ModelName is the Model Name that a user assigns to a model.

peType

Type is the kind of model. Currently, the only possible type is: data model.

peStatus

Status is the current state of the model.

Possible model states are:

M	Modifiable
W	Waiting for verify
C	Checked in
R	Read-only

U	Unusable

pszRelease

Release is the schema release for the model.

pszOwnerUserName

OwnerUserName is the user ID of the model owner.

pnModelCPage

ModelCPage is the Model Code Page.

pnModelLangCode

ModelLangCode is the Model Language Code.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchModelChkoutInfo
- EapiFetchModelCreateInfo
- EapiFetchModelLastUpdateInfo
- EApiFetchModelParentInfo

Example

```
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
NAME szModelName;
MODELTYPE eModelType;
MODELSTATUS eModelStatus;
RELEASE szRelease;
USERID szUserId;
CODEPAGE nCodePage;
LANGUAGECODE nLangCode;
nModelId = 9;
```

```
rc = EApiFetchModelInfo(nModelId,
szModelName,
&eModelType,
&eModelStatus,
szRelease,
szUserId,
&nCodePage,
&nLangCode);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchModelInfo\n");
else
{
    printf("ModelName = %s\n", szModelName);
    printf("ModelType = %c\n", (char)eModelType);
    printf("ModelStatus = %c\n", (char)eModelStatus);
    printf("Release = %s\n", szRelease);
    printf("UserId = %s\n", szUserId);
    printf("CodePage = %s\n", nCodePage);
    printf("LanguageCode = %s\n", nLangCode);
}
```

EApiFetchModelChkoutInfo

Description

Fetch Model Checkout Information. This function retrieves model checkout information from the encyclopedia. The information returned is checkout: ID, date, time, and user ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchModelChkoutInfo(	MODELID	nModelId,
	PCKOID	pnChkoutId,
	PDATE	pszDate,
	PTIME	pszTime,
	PUSERID	pszUserId);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model. This designates the specific model from which you want to retrieve the checkout information.

Output

pnChkoutId

The Checkout ID is a unique integer generated by the encyclopedia that identifies the model checkout.

pszDate

Date is the date of the checkout in year/month/day YYYY-MM-DD format.

pszTime

Time is the time the checkout was initiated in hours/minutes/seconds HH:MM:SS format on a 24 hour clock.

pszUserId

UserId is the user ID of the person who checked out the model.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_MODEL_NOT_CHECKED_OUT_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchModelInfo
- EapiFetchModelCreateInfo
- EapiFetchModelLastUpdateInfo
- EApiFetchModelParentInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
CKOID nChkoutId;
DATE szDate;
TIME szTime;
USERID szUserId;
nModelId = 9L;
rc = EApiFetchModelChkoutInfo(nModelId,
&nChkoutId,
szDate,
szTime,
szUserId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchChkoutInfo\n");
else
{
    printf("Checkout Id = %ld\n", nChkoutId);
    printf("Date = %s\n", szDate);
    printf("Time = %s\n", szTime);
    printf("UserId = %s\n", szUserId);

}
```

EApiFetchModelCreateInfo

Description

Fetch Model Create Information. This function retrieves model create information from the encyclopedia. The information returned is creation: date, time and user ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchModelCreateInfo(	MODELID	nModelId,
	PDATE	pszCDate,
	PTIME	pszCTime,
	PUSERID	pszCUserId),

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model. This designates the specific model from which you want to retrieve the creation information.

Output

pszCDate

CDate is the date the model was created in year/month/day YYYY-MM-DD format.

pszCTime

CTime is the time the model was created in hours/minutes/seconds HH:MM:SS format on a 24 hour clock.

pszCUserId

CUserId is the user ID of the person who created the model.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchModelInfo
- EapiFetchModelChkoutInfo
- EapiFetchModelLastUpdateInfo
- EApiFetchModelParentInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
DATE szDate;
TIME szTime;
USERID szUserId;
nModelId = 9L;
```

```
rc = EApiFetchModelCreateInfo(nModelId,
szDate,
szTime,
szUserId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchModelCreateInfo\n");
else
{
    printf("Date = %s\n", szDate);
    printf("Time = %s\n", szTime);
    printf("UserId = %s\n", szUserId);

}
```

EApiFetchModelLastUpdateInfo

Description

Fetch Model Last Update Information. This function retrieves model update information from the encyclopedia. The information returned is update: date, time and user ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchModelLastUpdateInfo(	MODELID	nModelId,
	PDATE	pszLUDate,
	PTIME	pszLUTime,
	PUSERID	pszLUUserId);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model. This designates the specific model from which you want to retrieve information on the latest updates.

Output

pszLUDate

LUDate is the date the model was last updated in year/month/day YYYY-MM-DD format.

pszLUTime

LUTime is the time the model was last updated in hours/minutes/seconds HH:MM:SS format on a 24 hour clock.

pszLUUserId

LUUserId is the user ID of the person who last updated the model.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchModelInfo
- EapiFetchModelChkoutInfo
- EapiFetchModelCreateInfo
- EApiFetchModelParentInfo

Example

```
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
DATE szDate;
TIME szTime;
USERID szUserId;
nModelId = 9L;
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchModelLastUpdateInfo\n");
```

```
else
{
    rc = EApiFetchModelLastUpdateInfo(nModelId,
    szDate,
    szTime,
    szUserId);
    printf("Date = %s\n", szDate);
    printf("Time = %s\n", szTime);
    printf("UserId = %s\n", szUserId);

}
```

EApiFetchModelParentInfo

Description

Fetch Model Parent Information. This function retrieves the input model parent information from the encyclopedia. The information returned is parent: encyclopedia ID, model ID, checkout ID, checkout date, and checkout time.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchModelParentInfo(	MODELID	nModelId,
	PENCYID	pnParentEncyid,
	PMODELID	pnParentModelid,
	PSUBSETID	pnParentCheckOutId,
	PDATE	pszParentCODate,
	PTIME	pszParentCOTime);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model. This designates the model from which you want to retrieve parent information.

Output

pnParentEncyid

ParentEncyid is the unique integer that identifies the Encyclopedia of the parent model.

pnParentModelid

ParentModelid is the unique integer that identifies the parent model.

pnParentCheckOutId

ParentCheckOutId is a unique integer generated by the encyclopedia that identifies the checkout of the parent model or subset. If the parent is in the Host Encyclopedia, the Parent Checkout ID is a Subset ID.

pszParentCODate

ParentCODate is the date the parent model was checked out in year/month/day YYYY-MM-DD format.

pszParentCOTime

ParentCOTime is the time the parent model was checked out in hours/minutes/seconds HH:MM:SS format on a 24 hour clock.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_MODEL_PARENT_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchModelInfo
- EapiFetchModelChkoutInfo
- EapiFetchModelCreateInfo
- EApiFetchModelLastUpdateInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nInModelId;
ENCYID nEncyId;
MODELID nOutModelId;
SUBSETID nSubsetId;
DATE szDate;
TIME szTime;
nInModelId = 9L;
rc = EApiFetchModelParentInfo(nInModelId,
&nEncyId,
&nOutModelId,
&nSubsetId,
szDate,
szTime);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchParentInfo\n");
else
{
    printf("EncyId = %s\n", nEncyId);
    printf("ModelId = %s\n", nOutModelId);
    printf("SubsetId = %s\n", nSubsetId);
    printf("Date = %s\n", szDate);
    printf("Time = %s\n", szTime);

}
```

EApiFetchModelByName

Description

Fetch Model by Name. Given a model name, this function retrieves its model ID. The information returned is the model ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchModelNameTypeObjs(	PNAME	pszModelName,
	PMODELID	pnModelId);

Input

pszModelName

ModelName is the string that identifies the Model.

Output

pnModelId

ModelID is a unique integer within the encyclopedia that identifies the model. This indicates the Model ID for the model specified name.

Return

EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

None.

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
NAME szModelName;
MODELID nModelId;
strcpy(szModelName, " CA Gen TEST MODEL");
rc = EApiFetchModelByName(szModelName,
&nModelId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchModelByName\n");
else
    printf("ModelId = %ld\n", nModelId);
```

Chapter 6: Model Lock and Unlock

This chapter describes the model lock and unlock subroutines.

This section contains the following topics:

EApiLockModel

Description

Lock Model locks the model designated by the model ID with its designated lock type. While a model is locked for read, no updates to the model are allowed by another process. This includes all encyclopedia processes. Only a value of EAPI_READ_LOCK and EAPI_WRITE_LOCK will be allowed for type of lock in this release. If updating models are stored in the CSE encyclopedia, the CSE servers must be running.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiLockModel(	MODELID	nModelId,
	ENCYLOCKTYPE	eLockType);

Input

MODELID

ModelId is a unique integer within the encyclopedia that identifies the model. This designates the model that you want to lock.

ENCYLOCKTYPE

LockType is the type of lock that will be applied to the model. Possible lock types are: Read or Write

A Write lock prevents access to the model. A Read lock prevents updates to the model.

EAPI_READ_LOCK = Read
EAPI_WRITE_LOCK = Write

Output

None.

Return

EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_MODEL_ALREADY_LOCKED_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

EApiUnLockModel

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
ENCYLOCKTYPE eLockType;
nModelId = 9L;
eLockType = EAPI_READ_LOCK;
rc = EApiLockModel(nModelId,
eLockType);
if ( rc != EAPI_SUCCESSFUL_RC )
printf("Error in EApiLockModel\n");
```

EApiUnLockModel

Description

Unlock Model unlocks the model designated by the model ID. This function requires the servers to be running in a client/server environment.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiUnLockModel(	MODELID	nModelId);

Input

ModelId

ModelID is a unique integer within the encyclopedia that identifies the model. This designates the specific model you want to unlock.

Output

None.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_LOCK_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

If you receive a return of *Model Not Found* or *Model Already Locked*, use the support client to release the lock before continuing. This situation might occur when you are running an application with the API, you lock the model, and then the API terminates abnormally (making it impossible to unlock the model using the API).

Related Functions

EApiLockModel

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
nModelId = 9L;
rc = EApiUnLockModel(nModelId);
if ( rc != EAPI_SUCCESSFUL_RC )
printf("Error in EApiUnLockModel\n");
```

Chapter 7: Object Information

This chapter describes the various functions to fetch the required object information. The Update API functions support adding objects to a model and deleting objects from a model.

This section contains the following topics:

EApiFetchAllModelObjects

Description

Fetch all Objects in a Model. This function retrieves all (or fewer) the object IDs in a specified model. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchAllModelObjects(	MODELID	nModelId,
	PLCOUNT	plNumObjects,
	POBJID	plArrayObjectIds);

Input

nModelId

ModelID is a unique integer within the encyclopedia that identifies the model. This designates the specific model from which you want to retrieve objects.

plCount

The count is the number of objects in the model. Specify the maximum number of object IDs that you want to retrieve. Specify negative one (-1) if you want to retrieve all the object IDs.

Output

plCount

This Count is the number of object IDs retrieved. The number of object IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

plArrayObjectIds

An Object ID is a unique identifier for an object within the encyclopedia. The array provides a set of Object IDs from the model.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiCountModelObjs

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
LCOUNT lCount, i;
OBJID nArrayObjIds[100];
nModelId = 9L;
lCount = 10;
rc = EApiFetchAllModelObjects(nInModelId,
&lCount,
nArrayObjIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchAllModelObjects\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("Object Id[%ld] = %ld\n", i, nArrayObjIds[i]);
}
```

EApiFetchAllModelTypeObjs

Description

Fetch all Objects of a given Type in a Model. This function retrieves all (or fewer) the object IDs of a certain object type in a specified model. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchAllModelTypeObjs(	MODELID	nModelId,
	OBJTYPECODE	nObjTypeCode,
	PLCOUNT	plNumObjects,
	POBJID	plArrayObjectIds);

Input

nModelId

ModelID is a unique integer within the encyclopedia that identifies the model.

nObjTypeCode

The Object Type Code identifies a particular object class. This value is used to identify all objects of the specified type. You can use the Object Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is otc.h.

plCount

The count is the number of objects of the specified type. Specify the maximum number of Object IDs that you want to retrieve. If you want to retrieve all of the Object IDs, specify negative one (-1).

Output

plCount

This count is the number of object IDs retrieved. The number of object IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

plArrayObjectIds

An Object ID is a unique identifier for an object within the encyclopedia. The array provides a set of Object IDs from the model.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_OTC_INVALID_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiCountModelTypeObjs

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
OBJTYPECODE nObjTypeCode;
LCOUNT lCount, i;
OBJID nArrayObjIds[100];
nModelId = 9L;
nObjTypeCode = 51;
lCount = 100L;
rc = EApiFetchAllModelTypeObjs(nModelId,
nObjTypeCode,
&lCount,
nArrayObjIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchAllTypeObjs\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("Object Id[%ld] = %ld\n", i, nArrayObjIds[i]);
}
```

EApiFetchModelNameTypeObjs

Description

Fetch Unique Object in Model by Name and Type. Given a model ID, object type code, property type code and object name, this function retrieves the object ID of the object with the given name. Use this function when the object name property is unique within the model.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchModelNameTypeObjs(	MODELID	nModelId,
	OBJTYPECODE	nObjTypeCode,
	PRPTYPECODE	nPrpTypeCode,
	PNAME	pszName,
	POBJID	pnObjectId);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model. This designates the specific model from which you want to retrieve Object IDs.

nObjTypeCode

An Object Type Code identifies a particular object class. This value is used to identify all objects of the specified type. You can use the Object Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is otc.h.

nPrpTypeCode

A Property Type Code is a unique identifier for a particular object property. This value indicates which name property from which to consider values. You can use the Property Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is ptc.h.

In this case, the input must be one of the following Property Type Codes:

- Name
- Loadname

pszName

Name Value

Output

pnObjectId

Object ID

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_OTC_INVALID_RC
EAPI_PTC_INVALID_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiFetchModelObjListByNameType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
OBJTYPECODE nObjTypeCode;
PRPTYPECODE nPrpTypeCode;
NAME szName;
OBJID nObjectId;
nModelId = 9L;
nObjTypeCode = 51;
nPrpTypeCode = 224;
strcpy(szName,"USES");
rc = EApiFetchModelNameTypeObjs(nModelId,
nObjTypeCode,
nPrpTypeCode,
szName,
&nObjectId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchModelNameTypeObjs\n");
else
    printf("ObjectId = %ld\n", nObjectId);
```

EApiFetchModelObjListByNameType

Description

Fetch List of Objects in Model by Name and Type. Given a model ID and object type code, property type code and object name, this function retrieves all (or fewer) object IDs. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchModelObjListByNameType(	MODELID	nModelId,
	OBJTYPECODE	nObjTypeCode,
	PRPTYPECODE	PrpTypeCode,
	PNAME	pszName,

	PLCOUNT	plCount,
	POBJID	plArrayObjectIds);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model. This designates the specific model from which you want to retrieve Object IDs.

nObjTypeCode

An Object Type Code identifies a particular object class. This value is used to identify all objects of the specified type. You can use the Object Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is otc.h.

nPrpTypeCode

A Property Type Code is a unique identifier for a particular object property. You can use the Property Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is ptc.h.

In this case, the input must be a Name Property Type Code.

pszName

Name is the Name Value.

plCount

The count is the number of objects of the specified name and type. Specify the maximum number of Object IDs you want to retrieve. If you want to retrieve all of the Object IDs, specify negative one (-1).

Output

plCount

This count is the number of objects of the specified name and type retrieved. The number of object IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

plArrayObjectIds

An Object ID is a unique identifier for an object within the encyclopedia. The array provides a set of Object IDs from the model.

In this case, the Object IDs retrieved are from the model specified, object type and property type.

Return

EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_OTC_INVALID_RC
EAPI_PTC_INVALID_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

- EapiCountModelNameTypeObjs
- EapiFetchModelNameTypeObjs

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
OBJTYPECODE nObjTypeCode;
PRPTYPECODE nPrpTypeCode;
NAME szName;
LCOUNT lCount, i;
OBJID nArrayObjIds[100];
nModelId = 9L;
nObjTypeCode = 51;
nPrpTypeCode = 224;
strcpy(szName,"CONTAINS");
lCount = 100L;
rc = EApiFetchModelObjListByNameType(nModelId,
nObjTypeCode,
nPrpTypeCode,
szName,
&lCount,
nArrayObjIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchModelObjListByNameType\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("Object Id[%ld] = %ld\n", i, nArrayObjIds[i]);

}
```

EApiFetchObjectInfo

Description

Fetch Object Information. Given an object ID, this function retrieves the object's model ID and object type code.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchObjectInfo(	OBJID	nObjectId,
	PMODELID	pnModelId,
	POBJTYPECODE	pnObjTypeCode);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the specified object.

Output

pnModelId

The Model ID is a unique integer within the encyclopedia that identifies the model. This indicates the specific model that contains the object.

pnObjTypeCode

The Object Type Code retrieved is for the Object ID specified.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchObjectCreateInfo
- EapiFetchObjectAncestryInfo
- EApiFetchObjChkoutParentInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObjectId;
MODELID nModelId;
OBJTYPECODE nObjTypeCode;
nObjectId = 74355L;
rc = EApiFetchObjectInfo(nObjectId,
&nModelId,
&nObjTypeCode);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchObjectInfo\n");
else
{
    printf("ModelId = %ld\n", nOutModelId);
    printf("ObjTypeCode = %ld\n", nObjTypeCode);

}
```

EApiFetchObjectCreateInfo

Description

Fetch Object Create Information. This function retrieves the input object ID information from the encyclopedia. The information returned is object: create date, create time and creation user ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchObjectCreateInfo(	OBJID	nObjectId,
	PDATE	pszCreateDate,
	PTIME	pszCreateTime,

	PUSERID	pnCreateUserIds);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the specified object.

Output

pszCreateDate

The date the object was created in year/month/day YYYY-MM-DD format.

pszCreateTime

The time the object was created in hours/minutes/seconds HH:MM:SS format on a 24 hour clock.

pnCreateUserIds

Create Userid is the user ID of the person who created the object.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchObjectInfo
- EapiFetchObjectAncestryInfo
- EApiFetchObjChkoutParentInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObjectId;
DATE szDate;
TIME szTime;
USERID szUserId;
nObjectId = 74344L;
rc = EApiFetchObjectCreateInfo(nObjectId,
szDate,
szTime,
szUserId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchObjectCreateInfo\n");
else
{
    printf("Date = %s\n", szDate);
    printf("Time = %s\n", szTime);
    printf("UserId = %s\n", szUserId);

}
```

EApiFetchObjectAncestryInfo

Description

Fetch Object Ancestry Information. This function retrieves the input object ID information from the encyclopedia. The information returned is object: original object ID and original encyclopedia ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchObjectAncestryInfo(	OBJID	nObjectId,
	PENCYID	pnEncyid,
	POBJID	pnObjectId);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the specified object.

Output

pnEncyid

Original Encyclopedia ID

pnObjectId

Original Object ID

Return

```
EAPI_SUCCESSFUL_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchObjectInfo
- EapiFetchObjectCreateInfo
- EapiFetchObjectWithAncestry
- EApiFetchObjChkoutParentInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nInObjectId;
ENCYID nEncyId,
OBJID nOutObjectId;
nInObjectId = 74344L;
```

```
rc = EApiFetchObjectAncestryInfo(nInObjectId,
&nEncyId,
&nOutObjectId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchObjectAncestryInfo\n");
else
{
    printf("EncyId = %ld\n", nEncyId);
    printf("Ancestor ObjectId = %ld\n", nOutObjectId);

}
```

EApiFetchObjectWithAncestry

Description

Fetch Object With Ancestry. This function retrieves the input object ID information from the encyclopedia.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchObjectWithAncestry(	MODELID	nModelId,
	ENCYID	nEncyid,
	OBJID	nObjectId,
	POBJID	pnObjectId);

Input

nModelId

ModelID is the model identification.

nEncyid

Encyid is the original Encyclopedia ID.

nObjectId

Original Object ID

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the specified object.

Output

pnObjectId

ObjectId is the ancestor object ID.

Return

EAPI_SUCCESSFUL_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

EApiFetchObjectAncestryInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
ENCYID nEncyId,
OBJID nInObjectId,
OBJID nOutObjectId;
nModelId = 9L;
nEncyId = 1L;
nInObjectId = 74344L;
rc = EApiFetchObjectWithAncestry(nModelId,
nEncyId,
nInObjectId,
&nOutObjectId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchObjectWithAncestry\n");
else
{
    printf("ObjectId = %ld\n", nOutObjectId);
}
```

EApiFetchObjChkoutParentInfo

Description

Fetch Object Checkout Parent Information. This function retrieves the input object ID information from the encyclopedia. The information returned is: parent object ID, maximum access and change status.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchObjChkoutParentInfo(	OBJID	nObjectId,
	POBJID	pnParentObjId,
	PMAXACCESS	pnMaxAccess,
	PCHGSTATUS	pnChangeStatus);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information on the specified object.

Output

pnParentObjId

ParentObjId is the Parent Object ID.

pnMaxAccess

MaxAccess is the Maximum Access.

pnChangeStatus

ChangeStatus is the Change Status.

Return

EAPI_SUCCESSFUL_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

- EapiFetchObjectInfo
- EapiFetchObjectCreateInfo
- EapiFetchObjectAncestryInfo
- EapiFetchObjChkoutParentInfo
- EapiFetchChkoutInfoForChkoutObj
- EapiFetchSubsetChkoutInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nInObjectId;
ENCYID nEncyId,
OBJID nOutObjectId;
nInObjectId = 74344L;
rc = EApiFetchObjChkoutParentInfo(nInObjectId,
&nOutObjectId,
&eMaxAccess,
&eChgStatus);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchObjChkoutParentInfo\n");
else
{
    printf("Parent ObjectId = %ld\n", nOutObjectId);
    printf("MaxAccess = %c\n", (char)eMaxAccess);
    printf("ChgStatus = %c\n", (char)eChgStatus);

}
```

EApiAddObject

Description

Add Object creates a new object within a particular model. This API will create and return the object id. The object will be created without a name.

Summary

EAPIRC EApiAddObject(	MODELID	nModelId,
	OBJTYPECODE	nObjTypeCode,
	OBJID	*pObjId);

Input

MODELID

The Model ID is a unique integer within the encyclopedia that identifies the model. This designates the specific model that you want to update.

OBJTYPECODE

An Object Type Code identifies a particular object class. This value designates the specific object type of the object you wish to add.

Output

OBJID

When the object is added to the model, this unique identifier is created for the object.

Return

```
EAPIRC returns the following information.
EAPI_SUCCESSFUL_RC if object added
EAPI_MODEL_NOT_LOCKED_RC if the model is not locked for update
EAPI_OBJECT_NOT_ADDED_RC if object not added to model
EAPI_MODEL_NOT_FOUND_RC if model not found
EAPI_OTC_INVALID_RC if object type code is invalid
EAPI_NOT_CONNECTED_RC if not connected to encyclopedia
EAPI_NOT_AUTHORIZED_RC if user is not authorized for add
EAPI_USER_NOT_LOGGED_ON_RC if user not logged on
```

Related Functions

EApiDelObject

Example

```
#include<eapidef.h>
MODELID modelid = 0;
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObj;
OBJTYPECODE nObjType = OTC_HLENT;
/* Adding new Entity type object to model */
rc = EApiAddObject(modelid,nObjType,&nObj);
if(rc != EAPI_SUCCESSFUL_RC)
    {
        printf("EApiAddObject is failed\n");
    }
    else
    {
        printf("Object id of new object is %ld\n",nObj);
    }
```

EApiDelObject

Description

Delete Object deletes an object from a particular model. A flag is passed to this API indicating whether or not to perform a trigger delete.

Summary

EAPIRC EApiDelObject(	OBJID	nObjId,
	char	trigger);

Input

OBJID

The unique identifier for the object you want to delete.

CHAR

The Trigger Delete Flag.

Y = delete object with triggers

Important! When the flag is set to *Y*, deleting an object or association will also delete other objects that depend on that object or association existing. This helps to ensure the integrity of the model.

N = delete object without triggers

Important! Use of the value *N* for the trigger delete flag may corrupt your model. If an object or association is deleted using this option, only that object or association is removed. None of the objects associated with the deleted object or association are removed.

An example of a potential corruption would be if an attribute is deleted with the *N* value for the trigger delete flag and if that attribute had views associated with it, the view objects would still exist and this would be invalid as there is no attribute for them to view. A similar situation would occur if the association between the entity and the attribute were deleted with the *N* value for the trigger delete flag. In this case, any views of the attribute would still exist but there would be no association to indicate that the attribute belongs to an entity.

Output

None.

Return

```
EAPIRC returns the following information.
EAPI_SUCCESSFUL_RC if object deleted
EAPI_MODEL_NOT_LOCKED_RC if the model is not locked for update
EAPI_OBJECT_PROTECTED_RC if the object is protected
EAPI_OBJECT_NOT_FOUND_RC if object not found
EAPI_NOT_CONNECTED_RC if not connected to encyclopedia
EAPI_NOT_AUTHORIZED_RC if user is not authorized for deletion
EAPI_USER_NOT_LOGGED_ON_RC if user is not logged on
```

Related Functions

EApiAddObject

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID       nObjectId;
CHARVALUE   trigger;
nObjectId = 87144L;
trigger = 'Y';
rc = EApiDelObject(nObjectId, trigger);
if ( rc != EAPI_SUCCESSFUL_RC )
printf("Error in EApiDelObject\n");
```

Chapter 8: Association Information

This chapter describes the various functions to fetch the association information of an object. The Update API functions support adding, deleting and reordering associations.

This section contains the following topics:

EApiFetchCardOneAsc

Description

Fetch Cardinality One Association. Given an object ID and single association type code, this function retrieves the association's destination object ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchCardOneAsc(	OBJID	nObjectId,
	ASCTYPECODE	nAscTypeCode,
	POBJID	pnDestObjectId);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the association occurrence.

nAscTypeCode

An Association Type Code is a unique identifier for an association between objects. This value is used to retrieve information about the specified association. You can use the Association Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is atc.h.

In this case, the input must be a Cardinality One Association Type Code.

Output

pnDestObjectId

DestObjectId is the Object ID of the Destination Object.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_SOURCE_OBJ_NOT_FOUND_RC
EAPI_DEST_OBJ_NOT_FOUND_RC
EAPI_ATC_INVALID_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchAscOccLastUpdInfo
- EapiFetchOrderAscOccInfo
- EapiFetchParentAscOccChkoutInfo
- EapiFetchCardManyAsc
- EApiFetchSchemaAscForObjType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObjectId;
ASCTYPECODE nAscTypeCode;
OBJID nDestObjectId;
nObjectId = 70900L;
nAscTypeCode = 186;
```

```
rc = EApiFetchCardOneAsc(nObjectId,
nAscTypeCode,
&nDestObjectId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchCardOneAsc\n");
else
    printf("DestObjId = %ld\n", nDestObjectId);
```

EApiFetchCardOneAscWithObjInfo

Description

Fetch Cardinality One Association. Given an object ID and single association type code, this function retrieves the association's destination object ID, object type code, and org object ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchCardOneAscWithObjInfo(	OBJID	nObjectId,
	ASCTYPECODE	nAscTypeCode,
	POBJID	pnDestObjectId,
	POBJTYPECODE	pnObjTypeCode,
	POBJID	pnOrgObjId);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the association occurrence.

nAscTypeCode

An Association Type Code is a unique identifier for an association between objects. This value is used to retrieve information about the specified association. You can use the Association Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is atc.h.

In this case, the input must be a Cardinality One Association Type Code.

Output

pnDestObjectId

DestObjectId is the Object ID of the Destination Object.

pnObjTypeCode

ObjTypeCode is the Object type code of the Destination Object.

pnOrgObjId

OrgObjId is the Original Object ID of the Destination Object.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_SOURCE_OBJ_NOT_FOUND_RC
EAPI_DEST_OBJ_NOT_FOUND_RC
EAPI_ATC_INVALID_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchAscOccLastUpdInfo
- EapiFetchOrderAscOccInfo
- EapiFetchParentAscOccChkoutInfo
- EapiFetchCardManyAsc
- EapiFetchCardManyAscWithObjInfo
- EApiFetchSchemaAscForObjType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObjectId;
ASCTYPECODE nAscTypeCode;
OBJID nDestObjectId;
OBJTYPECODE nObjTypeCode;
OBJID nOrgObjId;
nObjectId = 70900L;
nAscTypeCode = 186;
```

```
rc = EApiFetchCardOneAscWithObjInfo(nObjectId,
nAscTypeCode,
&nDestObjectId,
&nObjTypeCode,
&nOrgObjId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchCardOneAsc\n");

else
{
    printf("DestObjId = %ld\n", nDestObjectId);
    printf("ObjTypeCode = %d\n", nObjTypeCode);
    printf("OrgObjID = %ld\n, nOrgObjId);

}
```

EApiFetchCardManyAsc

Description

Fetch Cardinality Many Association. Given an object ID and cardinality many association type code, this function retrieves all (or fewer) association destination object IDs.

If the association is an ordered association, the destination object IDs will be returned in sequence number order. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchCardManyAsc(	OBJID	nObjectId,
	ASCTYPECODE	nAscTypeCode,
	PLCOUNT	plCount,
	POBJID	plArrayObjectIds);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve association occurrences.

nAscTypeCode

An Association Type Code is a unique identifier for an association between objects. This value is used to retrieve information on the specified association. You can use the Association Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is atc.h.

In this case, the input must be a Cardinality Many Association Type Code.

plCount

The count is the number of destination objects in the association. Specify the maximum number of object IDs you want to retrieve. If you want to retrieve all the object IDs, specify negative one (-1).

Output

plCount

This count is the number of object IDs retrieved. The number of object IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

plArrayObjectIds

An Object ID is a unique identifier for an object within the encyclopedia. The array provides a set of Object IDs in the association.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_SOURCE_OBJ_NOT_FOUND_RC
EAPI_ATC_INVALID_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiCountCardManyAsc
- EapiFetchOrderAscOccInfo

- EapiFetchAscOccLastUpdInfo
- EapiFetchParentAscOccChkoutInfo
- EApiFetchSchemaAscForObjType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
LCOUNT lCount, i;
OBJID nObjectId;
ASCTYPECODE nAscTypeCode;
OBJID nArrayObjIds[100];
nObjectId = 70900L;
nAscTypeCode = 175;
lCount = 100L;
rc = EApiFetchCardManyAsc(nObjectId,
nAscTypeCode,
&lCount,
nArrayObjIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchCardManyAsc\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("Object Id[%ld] = %ld\n", i, nArrayObjIds[i]);

}
```

EApiFetchCardManyAscWithObjInfo

Description

Fetch Cardinality Many Association. Given an object ID and cardinality many association type code, this function retrieves all (or fewer) association destination object IDs, and the object type code and org object ID for all destination objects.

If the association is an ordered association, the destination object IDs will be returned in sequence number order. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchCardManyAscWithObjInfo(	OBJID	nObjectId,
	ASCTYPECODE	nAscTypeCode,
	PLCOUNT	plCount,
	POBJID	plArrayObjectIds,
	POBJTYPECODE	pnArrayObjTypeCodes,
	POBJID	pnArrayOrgObjIds);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve association occurrences.

nAscTypeCode

An Association Type Code is a unique identifier for an association between objects. This value is used to retrieve information on the specified association. You can use the Association Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is atc.h.

In this case, the input must be a Cardinality Many Association Type Code.

plCount

The count is the number of destination objects in the association. Specify the maximum number of object IDs you want to retrieve. If you want to retrieve all the object IDs, specify negative one (-1).

Output

plCount

This count is the number of object IDs retrieved. The number of object IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

plArrayObjectIds

An Object ID is a unique identifier for an object within the encyclopedia. The array provides a set of Object IDs in the association.

pnArrayObjectTypeCodes

ArrayObjectTypeCodes are Object Type Codes for each object in plArrayObjectIds.

pnArrayOrgObjIds

ArrayOrgObjIds are Original Object IDs for each object returned in plArrayObjectIds.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_SOURCE_OBJ_NOT_FOUND_RC
EAPI_ATC_INVALID_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiCountCardManyAsc
- EapiFetchOrderAscOccInfo
- EapiFetchAscOccLastUpdInfo
- EapiFetchParentAscOccChkoutInfo
- EApiFetchSchemaAscForObjType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
LCOUNT lCount, i;
OBJID nObjectId;
ASCTYPECODE nAscTypeCode;
OBJID nArrayObjIds[100];
OBJTYPECODE nArrayObjTypeCodes[100];
OBJID nArrayOrgObjIds[100];
nObjectId = 70900L;
nAscTypeCode = 175;
lCount = 100L;
```

```
rc = EApiFetchCardManyAscWithObjInfo(nObjectId,
nAscTypeCode,
&lCount,
nArrayObjIds,
nArrayObjTypeCodes,
nArrayOrgObjIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchCardManyAsc\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("Object Id[%ld] = %ld\n", i, nArrayObjIds[i]);
    printf("Object Type Code[%Id] = %Id\n", i, nArrayObjTypeCodes[i]);
    printf("Original Object ID[%ld] = %lk\n", i, nArrayOrgObjIds[i]);

}
```

EApiFetchOrderAscOccInfo

Description

Fetch Ordered Association Occurrence Information. Given an object ID, association type code and destination object ID, this function retrieves the association sequence number and the next object ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchOrderAscOccInfo(	OBJID	nObjectId,
	ASCTYPECODE	nAscTypeCode,
	OBJID	nDestObjectId,
	PASCSEQ	pnSeqNum,
	POBJID	pnNextObjId);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information on the specified object.

Source Object ID

nAscTypeCode

An Association Type Code is a unique identifier for an association between objects. This value is used to retrieve information about the specified association. You can use the Association Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is atc.h.

In this case, the input can be any Association Type Code.

nDestObjectId

Destination Object ID

Output

pnSeqNum

SeqNum is the Sequence Number.

pnNextObjId

NextObjId is the Next Object ID (-1 if last association occurrence).

Return

```
EAPI_SUCCESSFUL_RC
EAPI_SOURCE_OBJ_NOT_FOUND_RC
EAPI_DEST_OBJ_NOT_FOUND_RC
EAPI_ASSOC_NOT_FOUND_RC
EAPI_ATC_INVALID_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchAscOccLastUpdInfo
- EapiFetchParentAscOccChkoutInfo
- EApiFetchSchemaAscForObjType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nInObjectId;
ASCTYPECODE nAscTypeCode;
OBJID nDestObjectId;
ASCSEQ nAscSeqNum;
OBJID nOutObjectId;
nInObjectId = 70900L;
nAscTypeCode = 186;
nDestObjectId = 70901L;
rc = EApiFetchOrderAscOccInfo(nInObjectId,
nAscTypeCode,
nDestObjectId,
nAscSeqNum,
nOutObjectId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchOrderAscOccInfo\n");
else
{
    printf("AscSeqNum = %ld\n", nAscSeqNum);
    printf("Next ObjectId = %ld\n", nOutObjectId);
}
```

EApiFetchAscOccLastUpdInfo

Description

Fetch Association Occurrence Last Update Information. Given an object ID, association type code and destination object ID, this function retrieves association occurrence information from the encyclopedia. The information returned is association last update: date, time and user ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchAscOccLastUpdInfo(	OBJID	nObjectId,
	ASCTYPECODE	nAscTypeCode,
	OBJID	nDestObjectId,
	PDATE	pszLUDate,

	PTIME	pszLUTime,
	PNAME	pnLUUserId);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the specified object.

Source Object ID

nAscTypeCode

An Association Type Code is a unique identifier for an association between objects. This value is used to retrieve information about the specified association. You can use the Association Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is atc.h.

In this case, the input must be a valid Association Type Code for the source object type.

nDestObjectId

DestObjectId is the Destination Object ID.

Output

pszLUDate

Last Update Date is the date of the update in year/month/day YYYY-MM-DD format.

pszLUTime

Last Update Time is the time the update was initiated in hours/minutes/seconds HH:MM:SS format on a 24 hour clock.

pnLUUserId

Last Update User ID is the user ID of the person who updated the model.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_SOURCE_OBJ_NOT_FOUND_RC
EAPI_DEST_OBJ_NOT_FOUND_RC
EAPI_ASSOC_NOT_FOUND_RC
EAPI_ATC_INVALID_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchOrderAscOccInfo
- EApiFetchSchemaAscForObjType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObjectId;
ASCTYPECODE nAscTypeCode;
OBJID nDestObjectId;
DATE szDate;
TIME szTime;
USERID szUserId;
nObjectId = 70900L;
nAscTypeCode = 186;
nDestObjectId = 70901L;
rc = EApiFetchAscOccLastUpdInfo(nObjectId,
nAscTypeCode,
nDestObjectId,
szDate,
szTime,
szUserId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchAscOccLastUpdInfo\n");
else
{
    printf("Date = %s\n", szDate);
    printf("Time = %s\n", szTime);
    printf("UserId = %s\n", szUserId);
}
```

EApiFetchParentAscOccChkoutInfo

Description

Fetch Association Occurrence Checkout from Parent Information. Given an object ID, association type code and destination object ID, this function retrieves association occurrence checkout information from the encyclopedia. The information returned is the association change status flag.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchParentAscOccChkoutInfo(	OBJID	nObjectId,
	ASCTYPECODE	nAscTypeCode,
	OBJID	nDestObjectId,
	PCHGSTATUS	eChgStatus);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the specified object.

Source Object ID

nAscTypeCode

An Association Type Code is a unique identifier for an association between objects. This value is used to retrieve information about the specified association. You can use the Association Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is atc.h.

In this case, the input can be any Association Type Code.

nDestObjectId

DestObjectId is the Destination Object ID.

Output

eChgStatus

ChgStatus is the Change Status Flag.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_SOURCE_OBJ_NOT_FOUND_RC
EAPI_DEST_OBJ_NOT_FOUND_RC
EAPI_ASSOC_NOT_FOUND_RC
EAPI_ATC_INVALID_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchAscOccLastUpdInfo
- EapiFetchOrderAscOccInfo
- EApiFetchSchemaAscForObjType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObjectId;
ASCTYPECODE nAscTypeCode;
OBJID nDestObjectId;
PCHGSTATUS eChgStatus;
nObjectId = 70900L;
nAscTypeCode = 186;
nDestObjectId = 70901L;
rc = EApiFetchParentAscOccChkoutInfo(nObjectId,
nAscTypeCode,
nDestObjectId,
&eChgStatus);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchParentAscOccChkoutInfo\n");

else
    printf("ChgStatus = %c\n", (char)eChgStatus);
```

EApiAddAssoc

Description

Add Association adds an association to a model. The association will be created between the From object ID and the To object ID.

Summary

EAPIRC EApiAddAssoc(	OBJID	nFromObjId,
	ASCTYPECODE	nATC,
	OBJID	nToObjId);

Input

OBJID

The unique identifier for the From object.

ASCTYPECODE

The Association Type Code is a unique identifier for an association between objects. This value is used to retrieve information about the specified association you want to add.

OBJID

The unique identifier for the To object.

Output

None.

Return

EAPIRC returns the following information.

```
EAPI_SUCCESSFUL_RC if association added
EAPI_MODEL_NOT_LOCKED_RC if the model is not locked for update
EAPI_FROM_TO_OBJECTS_NOT_IN_SAME_MODEL_RC if the from and to objects are not
contained in the same model
EAPI_FROM_OBJECT_NOT_FOUND_RC if from object is not found
EAPI_OBJECT_PROTECTED_RC if the object is protected
EAPI_TO_OBJECT_NOT_FOUND_RC if to object is not found
EAPI_ASSOC_ALREADY_EXISTS_RC if from object already has this association
EAPI_ATC_INVALID_RC if association type code is invalid or association type code is
not valid for input objects
EAPI_INVALID_CARDINALITY if cardinality is invalid; i.e. cardinality of one for the
input association and from object already has this association to another object
EAPI_NOT_CONNECTED_RC if not connected to encyclopedia
EAPI_NOT_AUTHORIZED_RC if user is not authorized for add
EAPI_USER_NOT_LOGGED_ON_RC if user is not logged on
```

Related Functions

- EapiDelAssoc
- EapiReordrAssocBefr
- EApiReordrAssocAftr

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID       nFromObjId;
OBJID       nToObjId;
ASCTYPECODE nAscTypeCode;
nFromObjId = 70900L;
nToObjId   = 80000L;
nAscTypeCode = 186;
rc = EApiAddAssoc(nFromObjId, nAscTypeCode,nToObjId);
if ( rc != EAPI_SUCCESSFUL_RC )
printf("Error in EApiAddAssoc\n");
```

EApiDelAssoc

Description

Delete Association deletes an association from a particular model. A flag is passed to this API indicating whether or not to perform a trigger delete.

Summary

EAPIRC EApiDelAssoc(	OBJID	nFromObjId,
	ASCTYPECODE	nATC,
	OBJID	nToObjId,
	CHARVALUE	trigger);

Input

OBJID

The unique identifier for the From object.

ASCTYPECODE

The Association Type Code is a unique identifier for an association between objects. This value is used to retrieve information about the specified association you want to delete.

OBJID

The unique identifier for the To object.

CHAR

The Trigger Delete Flag.

'Y' = delete object with triggers

Important! When the flag is set to 'Y', deleting an object or association will also delete other objects that depend on that object or association existing. This helps to ensure the integrity of the model.

'N' = delete object without triggers

Important! Use of the value 'N' for the trigger delete flag may corrupt your model. If an object or association is deleted using this option, only that object or association is removed. None of the objects associated with the deleted object or association are removed.

An example of a potential corruption would be if an attribute is deleted with the 'N' value for the trigger delete flag and if that attribute had views associated with it, the view objects would still exist and this would be invalid as there is no attribute for them to view. A similar situation would occur if the association between the entity and the attribute were deleted with the 'N' value for the trigger delete flag. In this case, any views of the attribute would still exist but there would be no association to indicate that the attribute belongs to an entity.

Output

None.

Return

EAPIRC returns the following information.

```
EAPI_SUCCESSFUL_RC if association deleted
EAPI_MODEL_NOT_LOCKED_RC if the model is not locked for update
EAPI_FROM_OBJECT_PROTECTED_RC if the from object is protected
EAPI_TO_OBJECT_PROTECTED_RC if the to object is protected
EAPI_FROM_OBJECT_NOT_FOUND_RC if from object is not found
EAPI_TO_OBJECT_NOT_FOUND_RC if to object is not found
EAPI_ASSOC_NOT_FOUND_RC if association is not found
EAPI_NOT_CONNECTED_RC if not connected to encyclopedia
EAPI_NOT_AUTHORIZED_RC if user is not authorized for deletion
EAPI_USER_NOT_LOGGED_ON_RC if user is not logged on
```

Related Functions

- EapiAddAssociation
- EapiReordrAssociationBefr
- EApiReordrAssociationAftr

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID       nFromObjId;
OBJID       nToObjId;
ASCTYPECODE nAscTypeCode;
CHARVALUE   trigger;
nFromObjId = 70900L;
nToObjId   = 80000L;
nAscTypeCode = 186;
trigger = 'Y';
rc = EApiDelAssoc(nFromObjId, nAscTypeCode, nToObjId, trigger);
if ( rc != EAPI_SUCCESSFUL_RC )
printf("Error in EApiDelAssoc\n");
```

EApiReordrAssocAftr

Description

Reorder Association - Move After moves Object 2 after Object 1. It will validate that there is an ordered association between the From object ID and objects 1 and 2.

Summary

EAPIRC EApiReordrAssocAftr(	OBJID	nFromObjId,
	ASCTYPECODE	nATC,
	OBJID	nObj1Id,
	OBJID	nObj2Id);

Input

OBJID

The unique identifier for the From object.

ASCTYPECODE

The Association Type Code is a unique identifier for an association between objects. This value is used to retrieve information about the specified association you want to add.

OBJID

The unique identifier for the Object 1 object ID.

OBJID

The unique identifier for the Object 2 object ID.

Output

None.

Return

EAPIRC returns the following information.

```
EAPI_SUCCESSFUL_RC if association reordered
EAPI_MODEL_NOT_LOCKED_RC if the model is not locked for update
EAPI_ASSOC1_PROTECTED_RC if association 1 is protected
EAPI_ASSOC2_PROTECTED_RC if association 2 is protected
EAPI_OBJECT_1_PROTECTED_RC if object 1 is protected
EAPI_OBJECT_2_PROTECTED_RC if object 2 is protected
EAPI_FROM_OBJECT_NOT_FOUND_RC if from object is not found
EAPI_OBJECT_1_NOT_FOUND_RC if object 1 is not found
EAPI_OBJECT_2_NOT_FOUND_RC if object 2 not found
EAPI_ASSOC1_NOT_FOUND_RC if association between From Object and Object 1 not found
EAPI_ASSOC2_NOT_FOUND_RC if association between From Object and Object 2 not found
EAPI_ASSOC_NOT_ORDERED_RC if association is not ordered
EAPI_NOT_CONNECTED_RC if not connected to encyclopedia
EAPI_NOT_AUTHORIZED_RC if user is not authorized for deletion
EAPI_USER_NOT_LOGGED_ON_RC if user is not logged on
```

Related Functions

- EapiAddAssociation
- EapiReordrAssociationBefr

- EApiDelAssoc

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID       nFromObjId;
OBJID       nObj1Id;
OBJID       nObj2Id;
ASCTYPECODE nAscTypeCode;
nFromObjId = 70900L;
nAscTypeCode = 186;
nObj1Id   = 80000L;
nObj2Id   = 80001L;
rc = EApiReordrAssocAftr(nFromObjId, nAscTypeCode, nObj1Id, nObj2Id);
if ( rc != EAPI_SUCCESSFUL_RC )
printf("Error in EApiReordrAssocAftr\n");
```

EApiReordrAssocBefr

Description

Reorder Association - Move Before moves Object 2 before Object 1. It will validate that there is an ordered association between the From object id and objects 1 and 2.

Summary

EAPIRC EApiReordrAssocBefr(	OBJID	nFromObjId,
	ASCTYPECODE	nATC,
	OBJID	nObj1Id,
	OBJID	nObj2Id);

Input

OBJID

The unique identifier for the From object.

ASCTYPECODE

The Association Type Code is a unique identifier for an association between objects. This value is used to retrieve information about the specified association you want to add.

OBJID

The unique identifier for the Object 1 object ID.

OBJID

The unique identifier for the Object 2 object ID.

Output

None.

Return

EAPIRC returns the following information.

```
EAPI_SUCCESSFUL_RC if association reordered
EAPI_MODEL_NOT_LOCKED_RC if the model is not locked for update
EAPI_FROM_OBJECT_PROTECTED_RC if the from object is protected
EAPI_OBJECT_1_PROTECTED_RC if object 1 is protected
EAPI_OBJECT_2_PROTECTED_RC if object 2 is protected
EAPI_FROM_OBJECT_NOT_FOUND_RC if from object is not found
EAPI_OBJECT_1_NOT_FOUND_RC if object 1 is not found
EAPI_OBJECT_2_NOT_FOUND_RC if object 2 not found
EAPI_ASSOC1_NOT_FOUND_RC if association between From Object and Object 1 not found
EAPI_ASSOC2_NOT_FOUND_RC if association between From Object and Object 2 not found
EAPI_ASSOC_NOT_ORDERED_RC if association is not ordered
EAPI_NOT_CONNECTED_RC if not connected to encyclopedia
EAPI_NOT_AUTHORIZED_RC if user is not authorized for deletion
EAPI_USER_NOT_LOGGED_ON_RC if user is not logged on
```

Related Functions

- EapiAddAssociation
- EapiReordrAssociationAftr
- EApiDelAssoc

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID       nFromObjId;
OBJID       nObj1Id;
OBJID       nObj2Id;
ASCTYPECODE nAscTypeCode;
nFromObjId = 70900L;
nAscTypeCode = 186;
nObj1Id   = 80000L;
nObj2Id   = 80001L;
rc = EApiReordrAssocBefr(nFromObjId, nAscTypeCode, nObj1Id, nObj2Id);
if ( rc != EAPI_SUCCESSFUL_RC )
printf("Error in EApiReordrAssocBefr\n");
```

Chapter 9: Property Information

This chapter describes the various functions to fetch the properties of an object. The Update API functions support updating property values.

This section contains the following topics:

EApiFetchSingleCharPrp

Description

Fetch Single Character Property. Given an object ID and property type code, this function retrieves the single character property.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSingleCharPrp(	OBJID	nObjectId,
	PRPTYPECODE	nPrpTypeCode,
	PCHARVALUE	pnCharPrp);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information on the specified object.

nPrpTypeCode

A Property Type Code is a unique identifier for a particular object property. This value is used to retrieve information on the specified property. You can use the Property Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is ptc.h.

In this case, the input must be a Character Property Type Code.

Output

pnCharPrp

CharPrp is the Character Property Value.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_PTC_INVALID_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchPrpLastUpInfo
- EapiFetchParentChkoutPrpInfo
- EapiFetchCharArrayPrp
- EApiFetchSchemaPrpForObjType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nInObjectId;
PRPTYPECODE nPrpTypeCode;
CHARVALUE cCharPrp;
nObjectId = 87144L;
nPrpTypeCode = 85;
```

```
rc = EApiFetchSingleCharPrp(nInObjectId,
nPrpTypeCode,
&cCharPrp);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchSingleCharPrp\n");
else
    printf("Character Prp = %c\n", (char)cCharPrp);
```

EApiFetchSingleNumericPrp

Description

Fetch Numeric Property. Given an object ID and property type code, this function retrieves a single numeric property.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSingleNumericPrp(	OBJID	nObjectId,
	PRPTYPECODE	nPrpTypeCode,
	PINTVALUE	pnIntPrp);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the specified object.

nPrpTypeCode

A Property Type Code is a unique identifier for a particular object property. This value is used to retrieve information on the specified property. You can use the Property Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is ptc.h.

In this case, the input must be numeric. This includes:

- short
- ushort
- long
- vlong

Output

pnIntPrp

IntPrp is the Numeric Property Value.

Return

EAPI_SUCCESSFUL_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_PTC_INVALID_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

- EapiFetchPrpLastUpInfo
- EapiFetchParentChkoutPrpInfo
- EapiFetchSchemaPrpForObjType
- EApiFetchSchemaIntPrpDflt

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObjectId;
PRPTYPECODE nPrpTypeCode;
INTVALUE nIntPrp;
nObjectId = 70902L;
nPrpTypeCode = 140;
rc = EApiFetchSingleNumericPrp(nObjectId,
nPrpTypeCode,
&nIntPrp);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchSingleNumericPrp\n");

else
    printf("IntPrp = %d\n", nIntPrp);
```

EApiFetchCharArrayPrp

Description

Fetch Character Array Property. Given an object ID and property type code, this function retrieves a character array property.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchCharArrayPrp(	OBJID	nObjectId,
	PRPTYPECODE	nPrpTypeCode,
	PTEXTVALUE	pszCharText);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information on the specified object.

nPrpTypeCode

A Property Type Code is a unique identifier for a particular object property. This value is used to retrieve information on the specified property. You can use the Property Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is ptc.h.

In this case, the input must be a character array of one of the following types:

- Name
- Load Name
- String
- Description

Output

pszCharText

CharText is the Character Array Property Value.

Return

EAPI_SUCCESSFUL_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_PTC_INVALID_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

- EapiFetchPrpLastUpInfo
- EapiFetchParentChkoutPrpInfo
- EapiFetchSingleCharPrp
- EApiFetchSchemaPrpForObjType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObjectId;
PRPTYPECODE nPrpTypeCode;
nObjectId = 70902L;
nPrpTypeCode = 224;
rc = EApiFetchCharArrayPrp(nInObjectId,
nPrpTypeCode,
szCharText);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchCharArrayPrp\n");
else
    printf("CharText = %s\n", szCharText);
```

EApiFetchPrpLastUpInfo

Description

Fetch Property Last Update Information. This function retrieves object property update information from the encyclopedia. The information returned is property last update: date, time and userid. Use this function when the object name property is not unique within the model.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchPrpLastUpInfo(	OBJID	nObjectId,
	PRPTYPECODE	nPrpTypeCode,
	PDATE	pszLUDate,
	PTIME	pszLUTime,
	PNAME	pnLUUserId);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the specified object.

nPrpTypeCode

A Property Type Code is a unique identifier for a particular object property. This value is used to retrieve information about the specified property. You can use the Property Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is ptc.h.

In this case, the input must be a valid Property Type Code for the object type.

Output

pszLUDate

LUDate, the Last Update Date, is the date of the update in year/month/day YYYY-MM-DD format.

pszLUTime

LUTime, the Last Update Time, is the time the update was initiated in hours/minutes/seconds HH:MM:SS format on a 24 hour clock.

pnLUUserId

LUUserId, the Last Update User ID, is the user ID of the person who updated the model.

Return

EAPI_SUCCESSFUL_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_PTC_INVALID_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

- EapiFetchParentChkoutPrpInfo
- EapiFetchSingleCharPrp
- EApiFetchSchemaPrpForObjType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObjectId;
PRPTYPECODE nPrpTypeCode;
DATE szDate;
TIME szTime;
USERID szUserId;
nObjectId = 74344L;
nPrpTypeCode = 224;
rc = EApiFetchPrpLastUpInfo(nObjectId,
nPrpTypeCode,
szDate,
szTime,
szUserId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchPrpLastUpInfo\n");

else
{
    printf("Date = %s\n", szDate);
    printf("Time = %s\n", szTime);
    printf("UserId = %s\n", szUserId);

}
```

EApiFetchParentChkoutPrpInfo

Description

Fetch Property Checkout from Parent Information. Given an object ID and property type code, this function retrieves the Property Change Status Flag.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchParentChkoutPrpInfo(	OBJID	nObjectId,
	PRPTYPECODE	nPrpTypeCode,
	PCHGSTATUS	eChgStatus);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information on the specified object.

nPrpTypeCode

A Property Type Code is a unique identifier for a particular object property. This value is used to retrieve information on the specified property. You can use the Property Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is ptc.h.

In this case, the input must be a valid Property Type Code for the object type.

Output

eChgStatus

ChgStatus is the Change Status Flag.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_PTC_INVALID_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchPrpLastUpInfo
- EApiFetchSchemaPrpForObjType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObjectId;
PRPTYPECODE nPrpTypeCode;
nObjectId = 74344L;
nPrpTypeCode = 224;
rc = EApiFetchParentChkoutPrpInfo(nObjectId,
nPrpTypeCode,
&eChgStatus);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchParentChkoutPrpInfo\n");
else
    printf("Change Status = %c\n", (char)eChgStatus);
```

EApiUpCharArray

Description

Update Character Array Property updates a character array property for a particular object id. Character set and character length rules are specified in the appendix titled *Object Name Validation Rules*.

Usage of special characters in the Label string, '(LABELSTR)', of the object causes generation failure. This limitation is applicable only for Proxy generation.

Important! If the name of the object must be unique within it's related parent, the name property will not be updated or added until the object is committed. An example of this is attributes which must be unique within the parent entity. Use the appendix titled *Object Name Validation Rules* to know which objects have names that are unique within a parent.

Summary

EAPIRC EApiUpCharArray(	OBJID	nObjId,
	PRPTYPECODE	nPTC,
	CHARVALUE	*pValue);

Input

OBJID

The unique identifier for the object for which you want to update the character array property.

PRPTYPECODE

A Property Type Code is a unique identifier for a particular object property. This value indicates which name property to consider values from.

CHARVALUE

CharValue is the Character Property Default Value of the character array property you want to update.

Output

None.

Return

EAPIRC returns the following information:

```
EAPI_SUCCESSFUL_RC if object found
EAPI_MODEL_NOT_LOCKED_RC if the model is not locked for update
EAPI_OBJECT_PROTECTED_RC if the object is protected
EAPI_OBJECT_NOT_FOUND_RC if object not found
EAPI_PTC_INVALID_RC if property type code is invalid, property type code is not valid
for input object, or property type code is not a character array property
EAPI_VALUE_INVALID_RC if length of value is invalid
EAPI_NOT_CONNECTED_RC if not connected to encyclopedia
EAPI_NOT_AUTHORIZED_RC if user is not authorized for updates
EAPI_USER_NOT_LOGGED_ON_RC if user is not logged on
```

Related Functions

- EapiUpNumeric
- EApiUpSingleChar

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
char szCharText[32] = "Vendor string";
OBJID       nObjectId;
PRPTYPECODE nPrpTypeCode;
nObjectId = 70902L;
nPrpTypeCode = 224;
rc = EApiUpCharArray(nObjectId, nPrpTypeCode, szCharText);
if ( rc != EAPI_SUCCESSFUL_RC )
printf("Error in EApiUpCharArray\n");
```

EApiUpNumeric

Description

Update Numeric Property updates a numeric property for a particular object ID.

Summary

EAPIRC EApiUpNumeric(	OBJID	nObjId,
	PRPTYPECODE	nPTC,
	INTVALUE	nValue);

Input

OBJID

The unique identifier for the object for which you want to update the numeric property.

PRPTYPECODE

A Property Type Code is a unique identifier for a particular object property. This value indicates which name property to consider values from.

INTVALUE

Value is the Integer Property Default Value.

Output

None.

Return

EAPIRC returns the following information:

```
EAPI_SUCCESSFUL_RC if object found
EAPI_MODEL_NOT_LOCKED_RC if the model is not locked for update
EAPI_OBJECT_PROTECTED_RC if the object is protected
EAPI_OBJECT_NOT_FOUND_RC if object not found
EAPI_PTC_INVALID_RC if property type code is invalid, property type code is not valid
for input object, or property type code is not an integer property
EAPI_NOT_CONNECTED_RC if not connected to encyclopedia
EAPI_NOT_AUTHORIZED_RC if user is not authorized for updates
EAPI_USER_NOT_LOGGED_ON_RC if user is not logged on
```

Related Functions

- EapiUpCharArray
- EApiUpSingleChar

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID       nObjectId;
PRPTYPECODE nPrpTypeCode;
INTVALUE    nIntPrp;
nObjectId = 70902L;
nPrpTypeCode = 224;
nIntPrp = 13;
rc = EApiUpNumeric(nObjectId, nPrpTypeCode, nIntPrp);
if ( rc != EAPI_SUCCESSFUL_RC )
printf("Error in EApiUpNumeric\n");
```

EApiUpSingleChar

Description

Update Single Character Property updates a single character property for a particular object id type code. All objects will be validated against the permitted values listed in the Metamodel. See the Orchestra documentation for details. If permitted values for a particular property type code are not in the metamodel, all values will be allowed.

Summary

EAPIRC EApiUpSingleChar(	OBJID	nObjId,
	PRPTYPECODE	nPTC,
	CHARVALUE	cValue);

Input

OBJID

The unique identifier for the object for which you want to update the single character.

PRPTYPECODE

A Property Type Code is a unique identifier for a particular object property. This value indicates which name property to consider values from.

CHARVALUE

CharValue is the Character Property Default Value of the single character you want to update.

Output

None.

Return

EAPIRC returns the following information:

```
EAPI_SUCCESSFUL_RC if object found
EAPI_MODEL_NOT_LOCKED_RC if the model is not locked for update
EAPI_OBJECT_PROTECTED_RC if the object is protected
EAPI_OBJECT_NOT_FOUND_RC if object not found
EAPI_PTC_INVALID_RC if property type code is invalid, property type code is not valid
for input object, or property type code is not a character property
EAPI_VALUE_INVALID_RC if property value is invalid
EAPI_NOT_CONNECTED_RC if not connected to encyclopedia
EAPI_NOT_AUTHORIZED_RC if user is not authorized for updates
EAPI_USER_NOT_LOGGED_ON_RC if user is not logged on
```

Related Functions

- EapiUpCharArray
- EApiUpNumeric

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID       nObjectId;
PRPTYPECODE nPrpTypeCode;
CHARVALUE   cCharPrp;
nObjectId = 87144L;
nPrpTypeCode = 397;
cCharPrp = 'N';
rc = EApiUpSingleChar(nObjectId, nPrpTypeCode, cCharPrp);
if ( rc != EAPI_SUCCESSFUL_RC )
printf("Error in EApiUpSingleChar\n");
```

Chapter 10: Subset Information

This chapter describes the various functions to fetch the subset information of an object.

This section contains the following topics:

EApiFetchSubsetsInModel

Description

Fetch Subsets within a Model. This function retrieves all (or fewer) of the subset IDs in the model specified. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSubsetsInModel(	MODELID	nModelId,
	PLCOUNT	plCount,
	PSUBSETID	plArraySubIds);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model. This designates the model from which you want to retrieve Subset IDs.

plCount

The count is the maximum number of Subset IDs you want to retrieve. Specify negative one (-1) if you want to retrieve all the Subset IDs.

Output

plCount

This count is the number of subsets retrieved. The number of subset IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

plArraySubIds

ArraySubIds is the Array of Subset IDs.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiCountSubsetsInModel

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
LCOUNT lCount, i;
SUBSETID nArraySubsetIds[100];
nModelId = 9L;
lCount = 100L;
```

```
rc = EApiFetchSubsetsInModel(nModelId,
&lCount,
nArraySubsetIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchSubsetsInModel\n");

else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("Subset Id[%ld] = %ld\n", i, nArraySubsetIds[i]);

}
```

EApiFetchSubsetInfo

Description

Fetch Subset Information. Given a subset ID, this function retrieves subset information from the encyclopedia. The information returned is subset: name, model ID, type and owner.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSubsetInfo(	SUBSETID	nSubsetId,
	PNAME	pszSubsetName,
	PMODELID	pnModelId,
	PSUBSETTYPE	pnSubsetType,
	USERID	pnOwnerId);

Input

nSubsetId

The Subset ID is a unique integer within the encyclopedia that identifies the subset. This indicates the subset from which to retrieve information.

Output

pszSubsetName

SubsetName is the Subset Name.

pnModelId

The Model ID is a unique integer within the encyclopedia that identifies the model.

pnSubsetType

SubsetType is the Subset Type.

pnOwnerId

OwnerId is the Subset Owner User ID.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_SUBSET_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchSubsetCreateInfo
- EApiFetchSubsetChkoutInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
SUBSETID nSubsetId;
NAME szSubsetName;
MODELID nModelId;
SUBSETTYPE eSubsetType;
USERID szUserId;
nSubsetId = 2L;
```

```
rc = EApiFetchSubsetInfo(nSubsetId,
szSubsetName,
&nModelId,
&eSubsetType,
szUserId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchSubsetInfo\n");
else
{
    printf("SubsetName = %s\n", szSubsetName);
    printf("ModelId = %ld\n", nModelId);
    printf("SubsetType = %c\n", (char)eSubsetType);
    printf("Owner = %s\n", szUserId);
}
```

EApiFetchSubsetCreateInfo

Description

Fetch Subset Create Information. Given a subset ID, this function retrieves subset information from the encyclopedia. The information returned is subset create: date, time and user ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSubsetCreateInfo(	SUBSETID	nSubsetId,
	PDATE	pszCreateDate,
	PTIME	pszCreateTime,
	PUSERID	pnCreateUserId);

Input

nSubsetId

The Subset ID is a unique integer within the encyclopedia that identifies the subset. This indicates the subset from which to retrieve information.

Output

pszCreateDate

Create Date is the date the subset was created in year/month/day YYYY-MM-DD format.

pszCreateTime

Create Time is the time the subset was created in hours/minutes/seconds HH:MM:SS format on a 24 hour clock.

pnCreateUserId

Create Userid is the user ID of the person who created the subset.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_SUBSET_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchSubsetInfo
- EApiFetchSubsetChkoutInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
SUBSETID nSubsetId;
DATE szDate;
TIME szTime;
USERID szUserId;
nSubsetId = 2L;
```

```
rc = EApiFetchSubsetCreateInfo(nSubsetId,
szDate,
szTime,
szUserId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchSubsetCreateInfo\n");
else
{
    printf("Date = %s\n", szDate);
    printf("Time = %s\n", szTime);
    printf("UserId = %s\n", szUserId);
}
```

EApiFetchSubsetByName

Description

Fetch Subset by Name. Given a model ID and a subset name, this function retrieves the corresponding subset ID from the encyclopedia.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSubsetByName(	MODELID	nModelId,
	PNAME	pszSubsetName,
	PSUBSETID	pnSubsetId);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model. This designates the model from which you want to retrieve a Subset ID.

pszSubsetName

SubsetName is the Subset Name.

Output

pnSubsetId

The Subset ID is a unique integer within the encyclopedia that identifies the subset. This is the ID for the specified Subset Name and Model ID.

Return

EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_SUBSET_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

None.

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
NAME szSubsetName;
SUBSETID nSubsetId;
nModelId = 9L;
strcpy(szSubsetName,"CA Gen TEST SUBSET");
rc = EApiFetchSubsetByName(nModelId,
szSubsetName,
&nSubsetId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchSubsetByName\n");
else
    printf("SubsetId = %ld\n", nSubsetId);
```

EApiFetchSubsetScopingObjs

Description

Fetch Scoping Objects in a Subset. This function retrieves all (or fewer) of the scoping object IDs in a specified subset. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSubsetScopingObjs(	SUBSETID	nSubsetId,
	PLCOUNT	plCount,
	POBJID	plArrayObjectIds);

Input

nSubsetId

The Subset ID is a unique integer within the encyclopedia that identifies the subset. This indicates the subset from which to retrieve Object IDs.

plCount

The count is the number of scoping objects. Specify the maximum number of scoping object IDs you want to retrieve. If you want to retrieve all the object IDs, specify negative one (-1).

Output

plCount

This count is the number of scoping objects within the subset. The number of object IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

plArrayObjectIds

An Object ID is a unique identifier for an object within the encyclopedia. The array provides a set of Object IDs from the model.

In this case, the Object IDs retrieved are scoping objects in the specified subset definition.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_SUBSET_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiCountSubsetScopingObjs
- EApiFetchSubsetScopingObjInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
SUBSETID nSubsetId;
LCOUNT lCount, i;
OBJID nArrayObjIds[100];
nSubsetId = 2L;
lCount = 100L;
rc= EApiFetchSubsetScopingObjs(nSubsetId,
&lCount,
nArrayObjIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchSubsetScopingObjs\n");

else

{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("Object Id[%ld] = %ld\n", i, nArrayObjIds[i]);

}
```

EApiFetchSubsetScopingObjInfo

Description

Fetch Scoping Object Information. Given a subset ID and scoping object ID, this function retrieves the scoping object expansion and the protection value.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSubsetScopingObjInfo(	SUBSETID	nSubsetId,
	OBJID	nObjectId,
	PSCOPEXP	pnExpValue,
	PSCOPPROT	pnProValue);

Input

nSubsetId

The Subset ID is a unique integer within the encyclopedia that identifies the subset. This indicates the subset from which to retrieve information.

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the specified scoping object.

Output

pnExpValue

ExpValue is the Expansion Value.

pnProValue

ProValue is the Protection Value.

Return

EAPI_SUCCESSFUL_RC
EAPI_SUBSET_NOT_FOUND_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_OBJECT_NOT_IN_SUBSET
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

- EapiFetchSubsetCreateInfo
- EapiFetchSubsetInfo
- EApiFetchSubsetChkoutInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
SUBSETID nSubsetId;
OBJID nObjectId;
SCOPEXP nExpValue;
SCOPPROT nProValue;
nSubsetId = 2L;
nObjectId = 74344L;

rc = EApiFetchSubsetScopingObjInfo(nSubsetId,
nObjectId,
&nExpValue,
&nProValue);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchSubsetScopingObjInfo\n");
else
{
    printf("ExpValue = %ld\n", (long)nExpValue);
    printf("ProValue = %ld\n", (long)nProValue);
}
```

EApiFetchSubsetChkoutInfo

Description

Fetch Subset Checkout Information. Given a subset ID, this function retrieves the subset checkout information. The information returned is subset checkout: ID, date, time and user ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSubsetChkoutInfo(	SUBSETID	nSubsetId,
	PCKOID	pnChkoutId,
	PDATE	pszCODate,
	PTIME	pszCOTime,
	PUSERID	pszCOUserId);

Input

nSubsetId

The Subset ID is a unique integer within the encyclopedia that identifies the subset. This indicates the subset from which to retrieve information.

Output

pnChkoutId

The Checkout ID is a unique integer generated by the encyclopedia that identifies the subset checkout.

pszCODate

CODate, the Checkout Date, is the date of the checkout in year/month/day YYYY-MM-DD format.

pszCOTime

Checkout Time is the time the checkout was initiated in hours/minutes/seconds HH:MM:SS format on a 24 hour clock.

pszCOUserId

COUserId, the Checkout Userid, is the user ID of the person who checked out the subset.

Return

EAPI_SUCCESSFUL_RC
EAPI_SUBSET_NOT_FOUND_RC
EAPI_SUBSET_NOT_CHECKED_OUT_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

- EapiFetchSubsetCreateInfo
- EapiFetchSubsetInfo
- EApiFetchSubsetScopingObjInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
SUBSETID nSubsetId;
CKOID nChkoutId;
DATE szDate;
TIME szTime;
USERID szUserId;
nSubsetId = 2L;
rc = EApiFetchSubsetChkoutInfo(nSubsetId,
&nChkoutId,
szDate,
szTime,
szUserId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchSubsetChkoutInfo\n");
else
{
    printf("Checkout Id = %ld\n", nChkoutId);
    printf("Date = %s\n", szDate);
    printf("Time = %s\n", szTime);
    printf("UserId = %s\n", szUserId);
}
```

Chapter 11: Checkout Information

This chapter describes the various functions to fetch the checkout information for an object.

This section contains the following topics:

EApiFetchChkoutsForObj

Description

Fetch Checkouts for an Object. Given a subset ID, this function retrieves all (or fewer) of the checkout IDs containing the specified object. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter API Variable Types. (see page 51)

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchChkoutsForObj(	OBJID	nObjectId,
	PLCOUNT	plCount,
	PCKOID	plArrayChkoutIds);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the specified object.

plCount

The count is the number of checkouts in which the object exists. Specify the maximum number of checkout IDs you want to retrieve. If you want to retrieve all the checkout IDs, specify negative one (-1).

Output

plCount

This count is the number of checkouts in which the object exists. The number of checkout IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

plArrayChkoutIds

ArrayChkoutIds is the Array of Checkout IDs (= Subset ID on Host Encyclopedia).

Return

```
EAPI_SUCCESSFUL_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiCountChkoutsForObj

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObjectId;
LCOUNT lCount, i;
CKOID nArrayCheckoutIds[100];
nObjectId = 74344L;
lCount = 100L;
```

```
rc = EApiFetchChkoutsForObj(nObjectId,
&lCount,
nArrayCheckoutIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchChkoutsForObj\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("Checkout Id[%ld] = %ld\n", i, nArrayCheckoutIds[i]);
}
```

EApiFetchChkoutInfoForChkoutObj

Description

Fetch Checkout Information for an Object in a Checkout. Given an object ID and checkout ID, this function retrieves the requested protection value and the granted protection value for that object.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchChkoutInfoForChkoutObj(	OBJID	nObjectId,
	CKOID	nChkoutId,
	PSCOPPROT	pnReqPVal,
	PCKOSTATUS	pnGrtPVal);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the specified object.

nChkoutId

The Checkout ID is a unique integer generated by the encyclopedia that identifies the model or subset checkout. This value is used to retrieve information on an object in a checkout.

Output

pnReqPVal

ReqPVal is the Requested Protection Value.

pnGrtPVal

GrtPVal is the Granted Protection Value.

Return

EAPI_SUCCESSFUL_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_CHECKOUT_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

None.

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObjectId;
CKOID nChkoutId;
SCOPPROT nReqPVal;
CKOSTATUS nGrtPVal;
nObjectId = 74344L;
nChkoutId = 65L;
rc = EApiFetchChkoutInfoForChkoutObj(nObjectId,
nChkoutId,
&nReqPVal,&nGrtPVal);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchChkoutInfoForChkoutObj\n");
else
{
    printf("ReqPVal = %ld\n", nReqPVal);
    printf("GrtPVal = %ld\n", nGrtPVal);
}
```

EApiFetchChkoutChkdOutObjs

Description

Fetch Checked Out Objects for a Checkout. Given a checkout ID, this function retrieves all (or fewer) of the object IDs in a specified checkout. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchChkoutChkdOutObjs(	CKOID	nChkoutId,
	PLCOUNT	plCount,
	POBJID	plArrayObjectIds);

Input

nChkoutId

The Checkout ID is a unique integer generated by the encyclopedia that identifies the model or subset checkout. This value is used to identify all objects in a checkout.

plCount

The count is the number of checked out objects. Specify the maximum number of Object IDs you want to retrieve. If you want to retrieve all the Object IDs, specify negative one (-1).

Output

plCount

This count is the number of objects in the specified checkout. The number of object IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

plArrayObjectIds

An Object ID is a unique identifier for an object within the encyclopedia. The array provides a set of Object IDs from the model.

In this case, the Object IDs retrieved are from the specified checkout.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_CHECKOUT_NOT_FOUND_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiCountChkoutChkdOutObjs

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
CKOID nChkoutId;
LCOUNT lCount, i;
OBJID nArrayObjIds[100];
nChkoutId = 65L;
lCount = 100L;
rc = EApiFetchChkoutChkdOutObjs(nChkoutId,
&lCount,
nArrayObjIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchChkoutChkdOutObjs\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("Object Id[%ld] = %ld\n", i, nArrayObjIds[i]);
}
```

EApiFetchChkoutSubsetModelId

Description

Fetch Model/Subset ID from Checkout ID. Given a checkout ID, this function retrieves the model ID and subset ID for that checkout.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchChkoutSubsetModelId(	CKOID	nChkoutId,
	PMODELID	pnModelId,
	PSUBSETID	pnSubsetId);

Input

nChkoutId

The Checkout ID is a unique integer generated by the encyclopedia that identifies the model and subset checkout. This value is used to retrieve the Model ID and Subset ID of the checkout.

Output

pnModelId

Model ID. If the subset is checked out, a negative one (-1) is returned in the Model ID field.

pnSubModelId

Subset ID. If the Model is checked out, a negative one (-1) is returned in the Subset ID field.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_CHECKOUT_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

None.

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
CKOID nChkoutId;
MODELID nModelSubsetId;
nChkoutId = 65L;
rc = EApiFetchChkoutSubsetModelId(nChkoutId,
&nModelSubsetId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchChkoutSubsetModelId\n");
else
    printf("Model\\Subset Id = %ld\n", nModelSubsetId);
```

Chapter 12: User and Group Information

This chapter describes the various functions to fetch the user and group information for an object.

This section contains the following topics:

EApiFetchAllUsers

Description

Fetch All Users. This function retrieves all (or fewer) of the user IDs in the encyclopedia. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchAllUsers(	PLCOUNT	plCount,
	PPUSERID	pszUserIds);

Input

plCount

The count is the number of users in the encyclopedia. Specify the maximum number of User IDs you want to retrieve. If you want to retrieve all of the User IDs, specify negative one (-1).

Output

plCount

This count is the number of user IDs retrieved. The number of user IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

pszUserIds

Array of User IDs

Return

EAPI_SUCCESSFUL_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

EApiCountAllUsers

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
LCOUNT lCount, i;
USERID szArrayUserIds[100];
lCount = 100L;
rc = EApiFetchAllUsers(lCount,
szArrayUserIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchAllUsers\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("User Id[%ld] = %s\n", i, nArrayUserIds[i]);
}
```

EApiFetchUserInfo

Description

Fetch User Information. Given a user ID, this function retrieves user name and authorization information. The information returned is user: name, create user authorization, and create model authorization and encyclopedia administrator flag.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchUserInfo(	PUSERID	szUserId,
	PUSERNAME	pszUserName,
	PADDUSRAUTH	peUserAuth,
	PADDMDLAUTH	peModelAuth,
	PUSERADMIN	peAdminAuth);

Input

szUserId

UserId is the User ID.

Output

pszUserName

UserName is the User Name.

peUserAuth

UserAuth is Create User Authorization.

peModelAuth

ModelAuth is Create Model Authorization.

peAdminAuth

AdminAuth is the Encyclopedia Administrator Flag.

Return

EAPI_SUCCESSFUL_RC
EAPI_USER_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

- EapiFetchUserCreateInfo
- EapiFetchGroupInfo
- EApiFetchGroupCreateInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
USERID szUserId;
USERNAME szUserName;
ADDUSRAUTH eUserAuth;
ADDMDLAUTH eModelAuth;
USERADMIN eAdminAuth;
strcpy(szUserId,"CHRIS");
rc = EApiFetchUserInfo(szUserId,
szUserName,
&eUserAuth,
&eModelAuth,
&eAdminAuth);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchUserInfo\n");
else
{
    printf("UserName = %s\n", szUserName);
    printf("UserAuth = %c\n", (char)eUserAuth );
    printf("ModelAuth = %c\n", (char)eModelAuth);
    printf("AdminAuth = %c\n", (char)eAdminAuth);
}
```

EApiFetchUserCreateInfo

Description

Fetch User Create Information. This function retrieves user creation information from the encyclopedia. The information returned is user creation: date, time, and user ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchUserCreateInfo(	PUSERID	szUserId,
	PDATE	pszCDate,
	PTIME	pszCTime,
	PUSERID	pszCUserId);

Input

szUserId

UserId is the User ID.

Output

pszCDate

CDate, Create Date, is the date the user was created in year/month/day YYYY-MM-DD format.

pszCTime

CTime, Create Time, is the time the user was created in hours/minutes/seconds HH:MM:SS format on a 24 hour clock.

pszCUserId

CUserId, Create User ID, is the user ID of the person who created the user.

Return

EAPI_SUCCESSFUL_RC
EAPI_USER_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

- EapiFetchUserInfo
- EapiFetchGroupInfo
- EApiFetchGroupCreateInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
USERID szUserId;
DATE szDate;
TIME szTime;
USERID szOutUserId;
strcpy(szUserId,"CHRIS");
rc = EApiFetchUserCreateInfo(szUserId,
szDate,
szTime,
szOutUserId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchUserCreateInfo\n");
else
{
    printf("Date = %s\n", szDate);
    printf("Time = %s\n", szTime);
    printf("UserId = %s\n", szOutUserId);
}
```

EApiFetchAllGroups

Description

Fetch All Groups. This function retrieves all (or fewer) of the group IDs in the encyclopedia. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types. (see page 51)

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchAllGroups(	PLCOUNT	plCount,
	PPGRPID	pszArrayGroupIds);

Input

plCount

The count is the number of groups in the encyclopedia. Specify the maximum number of Group IDs you want to retrieve. If you want to retrieve all of the Group IDs, specify negative one (-1).

Output

plCount

This count is the number of group IDs retrieved from the encyclopedia. The number of group IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

pszArrayGroupIds

ArrayGroupIds is the Array of Group IDs.

Return

EAPI_SUCCESSFUL_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

EApiCountAllGroups

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
LCOUNT lCount, i;
GRPID szArrayGroupIds[100];
lCount = 100L;
rc = EApiFetchAllGroups(&lCount,
szArrayGroupIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchAllGroups\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("Group Id[%ld] = %s\n", i, szArrayGroupIds[i]);
}
```

EApiFetchGroupInfo

Description

Fetch Group Information. This function retrieves the name of the specified group.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchGroupInfo(	PGRPID	szGroupId,
	PUSERNAME	pszUserName);

Input

szGroupId

GroupId is the Group ID.

Output

pszUserName

UserName is the Group Name.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_GROUP_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchGroupCreateInfo
- EapiFetchUserCreateInfo
- EApiFetchUserInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
GRPID szGroupId;
USERNAME szUserName;
strcpy(szGroupId,"ATEAM");
rc = EApiFetchGroupInfo(szGroupId,
szUserName);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchGroupInfo\n");
else
    printf("GroupName = %s\n", szUserName);
```

EApiFetchGroupCreateInfo

Description

Fetch Group Create Information. This function retrieves group creation information from the encyclopedia. The information returned is group creation: date, time and user ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchGroupCreateInfo(	PGRPID	szGroupId,
	PDATE	pszCDate,
	PTIME	pszCTime,
	PUSERID	pszCUserId);

Input

szGroupId

GroupId is the Group ID.

Output

pszCDate

CDate, Create Date, is the date the group was created in year/month/day YYYY-MM-DD format.

pszCTime

CTime, Create Time, is the time the group was created in hours/minutes/seconds HH:MM:SS format on a 24 hour clock.

pszCUserId

CUserId, Create User ID, is the user ID of the person who created the group.

Return

EAPI_SUCCESSFUL_RC
EAPI_GROUP_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

- EapiFetchGroupInfo
- EapiFetchUserInfo
- EApiFetchUserCreateInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
GRPID szGroupId;
DATE szDate;
TIME szTime;
USERID szUserId;
printf(szGroupId,"ATEAM");
rc = EApiFetchGroupCreateInfo(szGroupId,
szDate,
szTime,
szUserId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchGroupCreateInfo\n");
else
{
    printf("Date = %s\n", szDate);
    printf("Time = %s\n", szTime);
    printf("UserId = %s\n", szUserId);
}
```

EApiFetchUsersInGroup

Description

Fetch Users within a Group. This function retrieves all (or fewer) of the user IDs for a specified group. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchUsersInGroup(	PGRPID	szGroupId,
	PLCOUNT	plCount,
	PPUSERID	pszArrayUserIds);

Input

szGroupId

GroupId is the Group ID.

plCount

The count is the number of users in the group. Specify the maximum number of User IDs you want to retrieve. If you want to retrieve all of the User IDs, specify negative one (-1).

Output

plCount

This count is the number of user IDs retrieved from the group. The number of user IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

pszArrayUserIds

ArrayUserIds is the Array of User IDs.

Return

EAPI_SUCCESSFUL_RC
EAPI_GROUP_NOT_FOUND_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
EAPI_NOT_CONNECTED_RCEAPI_USER_NOT_LOGGED_ON_RC

Related Functions

EApiCountUsersInGroup

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
GRPID szGroupId;
LCOUNT lCount, i;
GRPID szArrayUserIds[100];
strcpy(szGroupId,"ATEAM");
lCount = 100L;
rc = EApiFetchUsersInGroup(szGroupId,
&lCount,
szArrayUserIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in FetchUsersInGroup\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("User Id[%ld] = %s\n", i, szArrayUserIds[i]);
}
```

EApiFetchGroupsUsersIn

Description

Fetch Groups User is Included In. This function retrieves all (or fewer) of the group IDs of which a specified user is a member. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchGroupsUsersIn(	PUSERID	szUserId,
	PLCOUNT	plCount,
	PPGRPID	pszArrayGroupIds);

Input

szUserId

UserId is the User ID.

plCount

The count is the number of groups in which the user exists. Specify the maximum number of group IDs you want to retrieve. If you want to retrieve all the group IDs, specify negative one (-1).

Output

plCount

This count is the number of group IDs retrieved. The number of group IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

pszArrayGroupIds

ArrayGroupIds is the Array of Group IDs.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_USER_NOT_FOUND_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiCountGroupUsersIn

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
USERID szUserId;
LCOUNT lCount, i;
GRPID szArrayGroupIds[100];
strcpy(szUserId,"CHRIS");
lCount = 100L;
rc = EApiFetchGroupsUsersIn(szUserId,
&lCount,
szArrayGroupIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchGroupsUsersIn\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("Group Id[%ld] = %s\n", i, szArrayGroupIds[i]);
}
```

Chapter 13: Authorization Information

This chapter describes the various functions to fetch the authorization information of users/ groups for a model.

This section contains the following topics:

EApiFetchUsersGrpsModelAuth

Description

Fetch Users/Groups Authorized for a Model. Given a model ID, this function retrieves all (or fewer) of the group/user IDs authorized for a specific model. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter API Variable Types. (see page 51)

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchUsersGrpsModelAuth(	MODELID	nModelId,
	PLCOUNT	plCount,
	PPUSERID	pszArrayUserIds);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model. This designates the model from which you want to retrieve.

plCount

The count is the number of users and groups authorized for a model. Specify the maximum number of user/group IDs that you want to retrieve. If you want to retrieve all the user and group IDs, specify negative one (-1).

Output

plCount

This count is the number of authorized users and groups retrieved from the model. The number of user and group IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

pszArrayUserIds

ArrayUserIds is the Array of User/Group IDs.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiCountUsersGrpsModelAuth

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
LCOUNT lCount, i;
GRPID szArrayUserIds;
nModelId = 9L;
lCount = 100L;
```

```
rc = EApiFetchUsersGrpsModelAuth(nModelId,
&lCount,
szArrayUserIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchUsersGrpsModelAuth\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("User Id[%ld] = %s\n", i, szArrayUserIds[i]);
}
```

EApiFetchModelAuthInfo

Description

Fetch Model Authorization Information. Given a model ID and user group ID, this function retrieves model authorization information. The information returned is model: update authorization, migrate authorization and generate authorization.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchModelAuthInfo(	MODELID	nModelId,
	USERID	pszUserId,
	PUPDATEAUTH	peUpdateAuth,
	PMIGRATEAUTH	peMigratAuth,
	PGENERATEAUTH	peGeneraAuth);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model.

pszUserId

UserId is the User/Group ID.

Output

peUpdateAuth

UpdateAuth is Update Authorization.

peMigratAuth

MigratAuth is Migrate Authorization.

peGeneraAuth

GeneraAuth is Generate Authorization.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_USER_NOT_FOUND_RC
EAPI_AUTH_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchModelAuthCreateInfo
- EApiFetchSubsetAuthCreateInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
USERID szUserId;
UPDATEAUTH eUpdateAuth,
MIGRATEAUTH eMigratAuth,
GENERATEAUTH eGeneraAuth;
nModelId = 9L;
strcpy(szUserId,"CHRIS");
```

```
rc = EApiFetchModelAuthInfo(nModelId,
szUserId,
eUpdateAuth,
eMigratAuth,
eGeneraAuth);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchModelAuthInfo\n");
else
{
    printf("Update Auth = %c\n", (char)eUpdateAuth);
    printf("Migrate Auth = %c\n", (char)eMigratAuth);
    printf("Generate Auth = %c\n", (char)eGeneraAuth);
}
```

EApiFetchModelAuthCreateInfo

Description

Fetch Model Authorization Create Information. Given a model ID and user/group ID, this function retrieves, authorization model create information. The information returned is model authorization create: date, time and user ID.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchModelAuthCreateInfo(	MODELID	nModelId,
	USERID	pszUserId,
	PDATE	pszCDate,
	PTIME	pszCTime,
	USERID	pszCUserId);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model.

pszUserId

User Id is the User/Group ID.

Output

pszCDate

CDate, Create Date, is the date the model authorization was created in year/month/day YYYY-MM-DD format.

pszCTime

CTime, Create Time, is the time the model authorization was created in hours/minutes/seconds HH:MM:SS format on a 24 hour clock.

pszCUserId

CUserId, Create User ID, is the user ID of the person who created model authorization.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_USER_NOT_FOUND_RC
EAPI_AUTH_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchModelAuthInfo
- EApiFetchSubsetAuthCreateInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
USERID szUserId;
DATE szDate;
TIME szTime;
USERID szOutUserId;
nModelId = 9L;
strcpy(szUserId,"CHRIS");
```

```
rc = EApiFetchModelAuthCreateInfo(nModelId,
szUserId,
szDate,
szTime,
szOutUserId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchModelAuthCreateInfo\n");
else
{
    printf("Date = %s\n", szDate);
    printf("Time = %s\n", szTime);
    printf("UserId = %s\n", szOutUserId);
}
```

EApiFetchUsersGrpsSubsetAuth

Description

Fetch Users/Groups Authorized for Subset. Given a subset ID, this function retrieves all (or fewer) of the group/user IDs authorized for the specific subset. Use the count parameter to set the maximum number of IDs to return.

Note: For more information, see Count and Array Functions in the chapter API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchUsersGrpsSubsetAuth(	SUBSETID	nSubsetId,
	PLCOUNT	plCount,
	PPUSERID	pszArrayUserIds);

Input

nSubsetId

The Subset ID is a unique integer within the encyclopedia that identifies the subset. This indicates the subset from which to retrieve information.

plCount

The count is the number of users and groups authorized for the subset. Specify the maximum number of user and group IDs you want to retrieve. If you want to retrieve all the user and group IDs, specify negative one (-1).

Output

plCount

This count is the number of authorized users and groups retrieved from the subset. The number of user or group IDs provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

pszArrayUserIds

ArrayUserIds is the Array of User/Group IDs.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_SUBSET_NOT_FOUND_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiCountUserGrpsSubsetAuth

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
SUBSETID nSubsetId;
LCOUNT lCount, i;
USERID szArrayUserIds[100];
nSubsetId = 2L;
lCount = 100L;
```

```
rc = EApiFetchUsersGrpsSubsetAuth(nSubsetId,
&lCount,
szArrayUserIds);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchUsersGrpsSubsetAuth\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("User/Group Id[%ld] = %s\n", i, szArrayUserIds[i]);
}
```

EApiFetchSubsetAuthCreateInfo

Description

Fetch Subset Authorization Create Information. Given a model ID and user ID, this function retrieves, subset authorization create information. The information returned is subset authorization create: date, time and userid.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSubsetAuthCreateInfo(	SUBSETID	nSubsetId,
	USERID	pszUserId,
	PDATE	pszCDate,
	PTIME	pszCTime,
	PUSERID	PszCUserId);

Input

nSubsetId

The Subset ID is a unique integer within the encyclopedia that identifies the subset. This indicates the subset from which to retrieve information.

pszUserId

UserId is the User/Group ID.

Output

pszCDate

CDate, Create Date, is the date the subset authorization was created in year/month/day YYYY-MM-DD format.

pszCTime

CTime, Create Time, is the time the subset authorization was created in hours/minutes/seconds HH:MM:SS format on a 24 hour clock.

pszCUserId

CUserId, Create User ID, is the user ID of the person who created subset authorization.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_SUBSET_NOT_FOUND_RC
EAPI_USER_NOT_FOUND_RC
EAPI_AUTH_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

- EapiFetchModelAuthInfo
- EApiFetchModelAuthCreateInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
SUBSETID nSubsetId;
USERID szUserId;
DATE szDate;
TIME szTime;
USERID szOutUserId;
nSubsetId = 2L;
strcpy(szUserId, "CHRIS");
```

```
rc = EApiFetchSubsetAuthCreateInfo(nSubsetId,
szUserId,
szDate,
szTime,
szOutUserId);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchSubsetAuthCreateInfo\n");
else
{
    printf("Date = %s\n", szDate);
    printf("Time = %s\n", szTime);
    printf("UserId = %s\n", szOutUserId);
}
```

Chapter 14: Schema Information

This chapter describes the various functions to fetch the schema information for an object.

This section contains the following topics:

EApiFetchSchemaObjTypeInfo

Description

Fetch Schema Object Type Code Information. Given an object type code, this function retrieves object type code information. The information returned is object mnemonic, aggregate boundary flag, toolset workstation physical structure, division type code and scoping object flag.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSchemaObjTypeInfo(	OBJTYPECODE	nObjTypeCode,
	PMNEMONIC	pszMnemonic,
	POBJAGGBND	peObjAggBnd,
	POBJPHYSSTR	pnObjPhysStr,
	PDIVTYPECODE	pnDivTypeCode,
	POBJSCOPING	peObjScoping);

Input

nObjTypeCode

An Object Type Code identifies a particular object class. This value is used to retrieve information on the object type.

Output

pszMnemonic

Mnemonic is the Object Mnemonic.

peObjAggBnd

ObjAggBnd is the Aggregate Boundary Flag.

pnObjPhysStr

ObjPhysStr is the Toolset Workstation Physical Structure.

pnDivTypeCode

DivTypeCode is the Division Type Code.

peObjScoping

ObjScoping is the Scoping Object Flag.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_OTC_INVALID_RC
```

Related Functions

- EapiFetchSchemaAscTypeCodeInfo
- EApiFetchSchemaPrpTypeInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJTYPECODE nObjTypeCode;
MNEMONIC szMnemonic;
OBJAGGBND eObjAggBnd;
OBJPHYSSTR nObjPhysStr;
DIVTYPECODE nDivTypeCode;
OBJSCOPING eObjScoping;
nObjTypeCode = 51;
rc = EApiFetchSchemaObjTypeInfo(nObjTypeCode,
szMnemonic,
&eObjAggBnd,
&nObjPhysStr,
&nDivTypeCode,
&eObjScoping);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchSchemaObjTypeInfo\n");
else
{
    printf("Mnemonic = %s\n", szMnemonic);
    printf("Object Agg. Bnd = %c\n", (char)eObjAggBnd);
    printf("Object Physical String = %d\n", nObjPhysStr);
    printf("Div. Type Code = %d\n", nDivTypeCode);
    printf("Object Scoping = %c\n", (char)eObjScoping);
}
```

EApiFetchSchemaObjTypeByMnem

Description

Fetch Schema Object Type Code by Mnemonic. Given an object mnemonic, this function retrieves the corresponding object type code.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSchemaObjTypeByMnem(	PMNEMONIC	pszMnemonic,
	POBJTYPECODE	pnObjTypeCode);

Input

pszMnemonic

Mnemonic is the Object Mnemonic.

Output

pnObjTypeCode

The Object Type Code retrieved is for the mnemonic specified.

Return

EAPI_SUCCESSFUL_RC
EAPI_MNEMONIC_INVALID_RC

Related Functions

None.

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MNEMONIC szMnemonic;
OBJTYPECODE nObjTypeCode;
strcpy(szMnemonic, "TECHSYS");
rc = EApiFetchSchemaObjTypeByMnem((PMNEMONIC) szMnemonic,
(POBJTYPECODE) &nObjTypeCode);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchSchemaObjTypeByMnem\n");
else
    printf("ObjTypeCode = %ld\n", nObjTypeCode);
```

EApiFetchSchemaPrpTypeInfo

Description

Fetch Schema Property Type Code Information. Given an object type code and property type code, this function retrieves property type code information. The information returned is property mnemonic, format, column and length.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSchemaPrpTypeInfo(	OBJTYPECODE	nObjTypeCode,
	PRPTYPECODE	nPrpTypeCode,
	PMNEMONIC	pszMnemonic,
	PPRPFORMAT	pePrpFormat,
	PPRPCOLUMN	pnPrpColumn,
	PPRPLENGTH	pnPrpLength);

Input

nObjTypeCode

An Object Type Code identifies a particular object class. This value is used to retrieve information on the object type.

nPrpTypeCode

A Property Type Code is a unique identifier for a particular object property. This value is used to retrieve information about the specified property. You can use the Property Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is ptc.h.

In this case, the input must be a valid Property Type Code for the object type code.

Output

pszMnemonic

Mnemonic is the Property Mnemonic.

pePrpFormat

PrpFormat is the Property Format.

pnPrpColumn

PrpColumn is the Property Column.

pnPrpLength

PrpLength is the Property Length.

Return

EAPI_SUCCESSFUL_RC
EAPI_OTC_INVALID_RC
EAPI_PTC_INVALID_RC

Related Functions

- EapiFetchSchemaAscTypeCodeInfo
- EApiFetchSchemaObjTypeInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJTYPECODE nObjTypeCode;
PRPTYPECODE nPrpTypeCode;
MNEMONIC szMnemonic;
PRPFORMAT ePrpFormat;
PRPCOLUMN nPrpColumn;
PRPLENGTH nPrpLength;
nObjTypeCode = 51;
nPrpTypeCode = 224;
rc = EApiFetchSchemaPrpTypeInfo((OBJTYPECODE) nObjTypeCode,
(PRPTYPECODE) nPrpTypeCode,
(PMNEMONIC) szMnemonic,
(PPRPFORMAT) &ePrpFormat,
(PPRPCOLUMN) &nPrpColumn,
(PPRPLENGTH) &nPrpLength);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchSchemaPrpTypeInfo\n");
else
{
    printf("Mnemonic = %s\n", szMnemonic);
    printf("PrpFormat = %c\n", (char)ePrpFormat);
    printf("PrpColumn = %ld\n", nPrpColumn);
    printf("PrpLength = %ld\n", nPrpLength);
}
```

EApiFetchSchemaCharPrpDflt

Description

Fetch Schema Character Property Default Value. Given an object type code and character property type code, this function retrieves the character property default value.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSchemaCharPrpDflt(	OBJTYPECODE	nObjTypeCode,
	PRPTYPECODE	nPrpTypeCode,
	PCHARVALUE	pcCharValue);

Input

nObjTypeCode

An Object Type Code identifies a particular object class. This value is used to retrieve information about the object type.

nPrpTypeCode

A Property Type Code is a unique identifier for a particular object property. This value is used to retrieve information on the specified property. You can use the Property Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is ptc.h.

In this case, the input must be a Character Property Type Code.

Output

pcCharValue

CharValue is the Character Property Default Value.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_OTC_INVALID_RC
EAPI_PTC_INVALID_RC
```

Related Functions

EApiFetchSingleCharPrp

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJTYPECODE nObjTypeCode;
PRPTYPECODE nPrpTypeCode;
CHARVALUE cCharPrp;
nObjTypeCode = 51;
nPrpTypeCode = 85;
rc = EApiFetchSchemaCharPrpDflt((OBJTYPECODE) nObjTypeCode,
(PRPTYPECODE) nPrpTypeCode,
(PCHARVALUE) &cCharPrp);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchSchemaCharPrpDflt\n");
else
    printf("Character Prp = %c\n", (char)cCharPrp);
```

EApiFetchSchemaIntPrpDflt

Description

Fetch Schema Integer Property Default Value. This function is an object type code and integer property type code that retrieves the integer property default value.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSchemaIntPrpDflt(	OBJTYPECODE	nObjTypeCode,
	PRPTYPECODE	nPrpTypeCode,
	PINTVALUE	pnIntValue);

Input

nObjTypeCode

An Object Type Code identifies a particular object class. This value is used to retrieve information about the object type.

nPrpTypeCode

A Property Type Code is a unique identifier for a particular object property. This value is used to retrieve information on the specified property. You can use the Property Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is ptc.h.

In this case, the input must be a Numeric Property Type Code.

Output

pnIntValue

IntValue is the Integer Property Default Value.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_OTC_INVALID_RC
EAPI_PTC_INVALID_RC
```

Related Functions

EApiFetchSingleNumericPrp

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJTYPECODE nObjTypeCode;
PRPTYPECODE nPrpTypeCode;
INTVALUE nIntPrp;
nObjTypeCode = 51;
nPrpTypeCode = 140;
```

```
rc = EApiFetchSchemaIntPrpDflt((OBJTYPECODE) nObjTypeCode,
(PRPTYPECODE) nPrpTypeCode,
(PINTVALUE) &nIntPrp);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchSchemaIntPrpDflt\n");
else
    printf("Integer Prp = %d\n", nIntPrp);
```

EApiFetchSchemaPrpForObjType

Description

Fetch All Schema Properties Valid for an Object Type. Given an object type code, this function retrieves all (or fewer) of the valid property type codes for that object type. Use the count parameter to set the maximum number of property type codes.

Note: For more information, see Count and Array Functions in the chapter API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSchemaPrpForObjType(	OBJTYPECODE	nObjTypeCode,
	PLCOUNT	plCount,
	PPRPTYPECODE	plArrayPrpCodes);

Input

nObjTypeCode

An Object Type Code identifies a particular object class. This value is used to retrieve information about the object type. You can use the Property Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is ptc.h.

plCount

The count is the number of property types. Specify the maximum number of property type codes that you want to retrieve. If you want to retrieve all the property type codes, specify negative one (-1).

Output

plCount

This count is the number of property types retrieved from the schema. The number of property type codes provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

plArrayPrpCodes

ArrayPrpCodes is the Array of Property Type Codes.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_OTC_INVALID_RC
EAPI_MAXIMUM_COUNT_REACHED_RC
```

Related Functions

- EapiCountSchemaPrpForObjType
- EapiFetchSingleCharPrp
- EapiFetchSingleNumericPrp
- EapiFetchCharArrayPrp
- EapiFetchPrpLastUpInfo
- EApiFetchParentChkoutPrpInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJTYPECODE nObjTypeCode;
LCOUNT lCount, i;
PPRPTYPECODE nArrayPrpCodes[100];
nObjTypeCode = 51;
lCount = 100L;
rc = EApiFetchSchemaPrpForObjType((OBJTYPECODE) nObjTypeCode,
(PLCOUNT) &lCount,
(PPRPTYPECODE) pnArrayPrpCodes);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchSchemaPrpForObjType\n");
else
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("Prop Code[%ld] = %ld\n", i, nArrayPrpCodes[i]);
}
```

EApiFetchSchemaAscTypeCodeInfo

Description

Fetch Schema Association Type Code Information. Given an object type code and association type code, this function retrieves association information. The information returned is association mnemonic, inverse type code, cardinality, order flag, modify referencing flag, direction flag, and aggregate action value.

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSchemaAscTypeCodeInfo(	OBJTYPECODE	nObjTypeCode,
	ASCTYPECODE	nAscTypeCode,
	PMNEMONIC	pszMnemonic,
	PASCTYPECODE	pnInverseAsc,
	PASCCARD	peAscCard,
	PASCORDER	peAscOrder,
	PASCMODREF	peAscModRef,

	PASCDIR	peAscDir,
	PASCAGGACT	peAscAggAct);

Input

nObjTypeCode

An Object Type Code identifies a particular object class. You can use the Object Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is otc.h.

nAscTypeCode

An Association Type Code is a unique identifier for an association between objects. This value is used to retrieve information on the specified association. You can use the Association Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is atc.h.

In this case, the input must be a valid Association Type Code for the object type.

Output

pszMnemonic

Mnemonic is the Association Mnemonic.

pnInverseAsc

InverseAsc is the Inverse Association Type Code.

peAscCard

AscCard is the Association Cardinality.

peAscOrder

AscOrder is the Association Order Flag.

peAscModRef

AscModRef is the Association Modifying/Referencing Flag.

peAscDir

AscDir is the Association Direction.

peAscAggAct

AscAggAct is the Association Aggregate Action Value.

Return

EAPI_SUCCESSFUL_RC
EAPI_OTC_INVALID_RC
EAPI_ATC_INVALID_RC

Related Functions

- EapiFetchSchemaObjTypeInfo
- EApiFetchSchemaPrpTypeInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJTYPECODE nObjTypeCode;
ASCTYPECODE nAscTypeCode;
MNEMONIC szMnemonic;
ASCTYPECODE nOutAscTypeCode;
ASCCARD eAscCard;
ASCORDER eAscOrder;
ASCMODREF eAscModRef;
ASCDIR eAscDir;
ASCAGGACT eAscAggAct;
nObjTypeCode = 51;
nAscTypeCode = 404;
rc = EApiFetchSchemaAscTypeCodeInfo((OBJTYPECODE) nObjTypeCode,
(ASCTYPECODE) nAscTypeCode,
(PMNEMONIC) szMnemonic,
(PASCTYPECODE) &nOutAscTypeCode,
(PASCCARD) &eAscCard,
(PASCORDER) &eAscOrder,
(PASCMODREF) &eAscModRef,
(PASCDIR) &eAscDir,
(PASCAGGACT) &eAscAggAct);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiFetchSchemaAscTypeCodeInfo\n");
else
{
    printf("Mnemonic = %s\n", szMnemonic);
    printf("Inverse AscTypeCode = %ld\n",nOutAscTypeCode);
    printf("AscCard = %c\n", (char)eAscCard);
    printf("AscOrder = %c\n", (char)eAscOrder);
    printf("AscModRef = %c\n",(char)eAscModRef);
    printf("AscDir = %c\n", (char)eAscDir);
    printf("AscAggAct = %c\n",(char)eAscAggAct);
}
```

EApiFetchSchemaAscForObjType

Description

Fetch All Schema Associations Valid for an Object Type. Given an object type code, this function retrieves all (or fewer) of the valid association codes for that object type. Use the count parameter to set the maximum number of association type codes.

Note: For more information, see Count and Array Functions in the chapter API Variable Types. (see page 51)

Summary

```
#include <eapidef.h>
```

EAPIRC EApiFetchSchemaAscForObjType(	OBJTYPECODE	nObjTypeCode,
	PLCOUNT	plCount,
	PASCTYPECODE	plArrayAscCodes);

Input

nObjTypeCode

An Object Type Code identifies a particular object class. This value is used to retrieve information about the object type. You can use the Object Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is otc.h.

plCount

The count is the number of association types. Specify the maximum number of association type codes you want to retrieve. If you want to retrieve all the association type codes, specify negative one (-1).

Output

plCount

This count is the number of association types retrieved from the schema. The number of association type codes provided in the output is equal to or less than the number specified in the input.

Important! You must allocate enough memory for the array to avoid writing to memory space that is not allocated.

plArrayAscCodes

AscCodes is the Array of Association Type Codes.

Return

EAPI_SUCCESSFUL_RC
EAPI_OTC_INVALID_RC
EAPI_MAXIMUM_COUNT_REACHED_RC

Related Functions

- EapiCountSchemaAscForObjType
- EapiFetchCardOneAsc
- EapiFetchCardManyAsc
- EapiFetchOrderAscOccInfo
- EapiFetchOccLastUpdInfo
- EApiFetchParentAscOccChkoutInfo

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJTYPECODE nObjTypeCode;
LCOUNT lCount, i;
PRPTYPECODE nArrayAscCodes[100];
nObjTypeCode = 51;
lCount = 100L;
rc = EApiFetchSchemaAscForObjType((OBJTYPECODE) nObjTypeCode,
(PLCOUNT) &lCount,
(PPRPTYPECODE) nArrayAscCodes);
if ( rc > EAPI_MAXIMUM_COUNT_REACHED_RC)
    printf("Error in EApiFetchSchemaAscForObjType\n");
else
{
    printf("Count = %ld\n", lCount);
    for (i=0; i < lCount; i++)
    printf("Association Code[%ld] = %ld\n", i, nArrayAscCodes[i]);
}
```

Chapter 15: Count Information

This chapter describes the various functions to fetch the count information for a model.

This section contains the following topics:

EApiCountEncyModelIds

Description

Count all Models in the Encyclopedia. This function returns a count of all models in the encyclopedia. This function can be used to determine the maximum array size for retrieval of the model IDs.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountEncyModelIds(	PLCOUNT	plCount);

Input

None.

Output

plCount

The count is the number of Models in the Encyclopedia.

Return

EAPI_SUCCESSFUL_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

EApiFetchEncyModelIds

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
LCOUNT lCount;
rc = EApiCountEncyModelIds((PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountEncyModelIds\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountModelObjs

Description

Count all Objects in a Model. This function returns a count of all objects in a specified model. This function can be used to determine the maximum array size for retrieval of the object IDs.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountModelTypeObjs(	MODELID	nModelId,
	PLCOUNT	plCount);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model.

Output

plCount

The Count is the number of objects in the model specified.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiFetchModelObjs

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
LCOUNT lCount;
nModelId = 9L;
rc = EApiCountModelObjs((MODELID) nModelId,
(PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountModelObjs\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountModelTypeObjs

Description

Count all Objects of a given Type in a Model. This function returns a count of all objects in a specified model with a certain object type. This function can be used to determine the maximum array size for retrieval of the object IDs.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountModelTypeObjs(	MODELID	nModelId,
	OBJTYPECODE	nObjTypeCode,
	PLCOUNT	plCount);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model.

nObjTypeCode

An Object Type Code identifies a particular object class. This value is used to identify all objects of the specified type. You can use the Object Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is otc.h.

Output

plCount

The Count is the number of objects in the Model of the specified type.

Return

EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_OTC_INVALID_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

EApiFetchModelTypeObjs

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
OBJTYPECODE nObjTypeCode;
LCOUNT lCount;
nModelId = 9L;
nObjTypeCode = 51;
rc = EApiCountModelTypeObjs((MODELID) nModelId,
(OBJTYPECODE) nObjTypeCode,
(PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountModelTypeObjs\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountModelNameTypeObjs

Description

Count Number of Objects in Model by Name and Type. This function returns a count of all objects in a specified model with a certain object type and name. This function can be used to determine the maximum array size for retrieval of the object IDs.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountModelNameTypeObjs(	MODELID	nModelId,
	OBJTYPECODE	nObjTypeCode,
	PRPTYPECODE	nPrpTypeCode,
	PNAME	szName,
	PLCOUNT	plCount);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model.

nObjTypeCode

An Object Type Code identifies a particular object class. This value is used to identify all objects of the specified type. You can use the Object Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is otc.h.

nPrpTypeCode

A Property Type Code is a unique identifier for a particular object property. This value is used to retrieve information on the specified property. You can use the Property Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is ptc.h.

In this case, the input must be a Name Property Type Code.

szName

Name is the Name Value.

Output

plCount

The Count is the number of objects in the model of the specified name and types.

Return

EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_OTC_INVALID_RC
EAPI_PTC_INVALID_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

EApiFetchModelObjListByNameType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
OBJTYPECODE nObjTypeCode;
PRPTYPECODE nPrpTypeCode;
NAME szName;
LCOUNT lCount;
nModelId = 9L;
nObjTypeCode = 51;
nPrpTypeCode = 224;
strcpy(szName,"CONTAINS");
rc = EApiCountModelNameTypeObjs((MODELID) nModelId,
(OBJTYPECODE) nObjTypeCode,
(PRPTYPECODE) nPrpTypeCode,
(PNAME) szName,
(PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountModelNameTypeObjs\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountSubsetScopingObjs

Description

Count Scoping Objects in a Subset. This function returns a count of all scoping objects in a specified subset. This function can be used to determine the maximum array size for retrieval of the scoping object IDs.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountSubsetScopingObjs(	SUBSETID	nSubsetId,
	PLCOUNT	plCount);

Input

nSubsetId

The Subset ID is a unique integer within the encyclopedia that identifies the subset. This indicates the subset from which to retrieve Object IDs.

Output

plCount

The Count is the number of Scoping Objects in the Subset.

Return

EAPI_SUCCESSFUL_RC
EAPI_SUBSET_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

EApiFetchSubsetScopingObjs

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
SUBSETID nSubsetId;
LCOUNT lCount;
nSubsetId = 2L;
rc = EApiCountSubsetScopingObjs((SUBSETID) nSubsetId,
(PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountSubsetScopingObjs\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountChkoutChkdOutObjs

Description

Count Checked Out Objects for a Checkout. This function returns a count of all objects checked out in a specified checkout. This function can be used to determine the maximum array size for retrieval of the object IDs.

Note: For more information, see Count and Array Functions in the chapter title API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountChkoutChkdOutObjs(	CKOID	nChkoutId,
	PLCOUNT	plCount);

Input

nChkoutId

The Checkout ID is a unique integer generated by the encyclopedia that identifies the model or subset checkout. This value is used to identify all objects in a checkout.

Output

plCount

Count is the number of objects checked out in the specified checkout.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_CHECKOUT_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiFetchChkoutChkdOutObjs

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
SUBSETID nChkoutId;
LCOUNT lCount;
nChkoutId = 65L;
rc = EApiCountChkoutChkdOutObjs((SUBSETID) nChkoutId,
(PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountChkoutChkdOutObjs\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountSubsetsInModel

Description

Count Subsets within a Model. This function returns a count of all subset IDs in a specified model. This function can be used to determine the maximum array size for retrieval of the subset IDs.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountSubsetsInModel(	MODELID	nModelId,
	PLCOUNT	plCount);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model.

Output

plCount

Count is the number of subsets in the model specified.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiFetchSubsetsInModel

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
LCOUNT lCount;
nModelId = 9L;
rc = EApiCountSubsetsInModel((MODELID) nModelId,
(PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountSubsetsInModel\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountChkoutsForObj

Description

Count Checkouts for an Object. This function returns a count of all checkouts for a given object ID. This function can be used to determine the maximum array size for retrieval of the checkout IDs.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountChkoutsForObj(	OBJID	nObjectId,
	PLCOUNT	plCount);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the specified object.

Output

plCount

Count is the number of checkouts in which the specified object is located.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_OBJECT_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiFetchChkoutsForObj

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObjectId;
LCOUNT lCount;
nObjectId = 74344L;
rc = EApiCountChkoutsForObj((OBJID) nObjectId,
(PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountChkoutsForObj\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountCardManyAsc

Description

Count Cardinality Many Association. This function returns a count of all association destination object IDs for a given object ID and a given association type. This function can be used to determine the maximum array size for retrieval of the association destination object IDs.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types. (see page 51)

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountCardManyAsc(	OBJID	nObjectId,
	ASCTYPECODE	nAscTypeCode,
	PLCOUNT	plCount);

Input

nObjectId

An Object ID is a unique identifier for an object within an encyclopedia. This value is used to retrieve information about the specified object.

nAscTypeCode

An Association Type Code is a unique identifier for an association between objects. This value is used to retrieve information about the specified association. You can use the Association Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is atc.h.

In this case, the input must be a Cardinality Many Association Type Code.

Output

plCount

The Count is the number of destination objects retrieved of the association type specified.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_SOURCE_OBJ_NOT_FOUND_RC
EAPI_ATC_INVALID_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiFetchCardManyAsc

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJID nObjectId;
ASCTYPECODE nAscTypeCode;
LCOUNT lCount;
nObjectId = 70900L;
nAscTypeCode = 175;
rc = EApiCountCardManyAsc((OBJID) nObjectId,
(ASCTYPECODE) nAscTypeCode,
(PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountCardManyAsc\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountAllUsers

Description

Count All Users. This function returns a count of all users in the encyclopedia. This function can be used to determine the maximum array size for retrieval of the user IDs.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountAllUsers(	PLCOUNT	plCount);

Input

None.

Output

plCount

The Count is the number of users in the Encyclopedia.

Return

EAPI_SUCCESSFUL_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

EApiFetchAllUsers

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
LCOUNT lCount;
rc = EApiCountAllUsers((PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountAllUsers\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountAllGroups

Description

Count All Groups. This function returns a count of all groups in the encyclopedia. This function can be used to determine the maximum array size for retrieval of the group IDs.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountAllGroups(	PLCOUNT	plCount);

Input

None.

Output

plCount

The Count is the number of groups in the Encyclopedia.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiFetchAllGroups

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
LCOUNT lCount;
rc = EApiCountAllGroups((PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountAllGroups\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountUsersInGroup

Description

Count Users within a Group. This function returns a count of all user IDs in a specified group ID. This function can be used to determine the maximum array size for retrieval of the user IDs.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountUsersInGroup(	PGRPID	szGroupId,
	PLCOUNT	plCount);

Input

szGroupId

GroupId is the Group ID.

Output

plCount

The Count is the number of users in the Group specified.

Return

EAPI_SUCCESSFUL_RC
EAPI_GROUP_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC

Related Functions

EApiFetchUsersInGroup

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
GRPID szGroupId;
LCOUNT lCount;
strcpy(szGroupId, "ATEAM");
rc = EApiCountUsersInGroup((PGRPID) szGroupId,
(PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountUsersInGroup\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountGroupsInUser

Description

Count Groups User is Included In. This function returns a count of all group IDs that have the specified user ID as a member. This function can be used to determine the maximum array size for retrieval of the group IDs.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountGroupsInUser(	PUSERID	szUserId,
	PLCOUNT	plCount);

Input

szUserId

UserId is the user ID.

Output

plCount

The Count is the number of groups in which the user is included.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_USER_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiFetchGroupsInUser

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
USERID szUserId;
LCOUNT lCount;
strcpy(szUserId, "CHRIS");
rc = EApiCountGroupsInUser((PUSERID) szUserId,
(PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountGroupsInUser\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountUsersGrpsModelAuth

Description

Count Users/Groups Authorized for a Model. This function returns a count of all user/group IDs which are authorized for the specified model. This function can be used to determine the maximum array size for retrieval of the user/group IDs.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types. (see page 51)

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountUsersGrpsModelAuth(	MODELID	nModelId,
	PLCOUNT	plCount);

Input

nModelId

The Model ID is a unique integer within the encyclopedia that identifies the model. This indicates the model from which to retrieve User/Group IDs.

Output

plCount

The Count is the number of users that are authorized for the model specified.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_MODEL_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiFetchUsersGrpsModelAuth

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
MODELID nModelId;
LCOUNT lCount;
nModelId = 9L;
rc = EApiCountUsersGrpsModelAuth((MODELID) nModelId,
(PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountUsersGrpsModelAuth\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountUsersGrpsSubsetAuth

Description

Count Users/Groups Authorized for Subset. This function returns a count of all user/group IDs authorized for the specified subset. This function can be used to determine the maximum array size for retrieval of the user/group IDs.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountUsersGrpsSubsetAuth(	SUBSETID	nSubsetId,
	PLCOUNT	plCount);

Input

nSubsetId

The Subset ID is a unique integer within the encyclopedia that identifies the subset. This indicates the subset from which to retrieve User/Group IDs.

Output

plCount

The Count is the number of users and groups authorized for the subset specified.

Return

```
EAPI_SUCCESSFUL_RC
EAPI_SUBSET_NOT_FOUND_RC
EAPI_NOT_CONNECTED_RC
EAPI_USER_NOT_LOGGED_ON_RC
```

Related Functions

EApiFetchUsersGrpsSubsetAuth

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
SUBSETID nSubsetId;
LCOUNT lCount;
nSubsetId = 2L;
rc = EApiCountUsersGrpsSubsetAuth((SUBSETID) nSubsetId,
(PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountUsersGrpsSubsetAuth\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountSchemaPrpForObjType

Description

Count Schema Properties for Object Type. This function returns a count of all properties for a specified object type. This function can be used to determine the maximum array size for retrieval of the property types.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types (see page 51).

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountSchemaPrpForObjType(	OBJTYPECODE	nObjTypeCode,
	PLCOUNT	plCount);

Input

nObjTypeCode

An Object Type Code identifies a particular object class. This value is used to retrieve information on the specified object type. You can use the Object Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is otc.h.

Output

plCount

The Count is the number of properties of the specified object type.

Return

EAPI_SUCCESSFUL_RC
EAPI_OTC_INVALID_RC

Related Functions

EApiFetchSchemaPrpForObjType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJTYPECODE nObjTypeCode;
LCOUNT lCount;
nObjTypeCode = 51;
rc = EApiCountSchemaPrpForObjType((OBJTYPECODE) nObjTypeCode,
(PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountSchemaPrpForObjType\n");
else
    printf("Count = %ld\n", lCount);
```

EApiCountSchemaAscForObjType

Description

Count Schema Associations for Object Type. This function returns a count of all associations for a specified object type. This function can be used to determine the maximum array size for retrieval of the association types.

Note: For more information, see Count and Array Functions in the chapter titled API Variable Types. (see page 51)

Summary

```
#include <eapidef.h>
```

EAPIRC EApiCountSchemaAscForObjType(	OBJTYPECODE	nObjTypeCode,
	PLCOUNT	plCount);

Input

nObjTypeCode

An Object Type Code identifies a particular object class. This value is used to retrieve information about the specified object type. You can use the Object Type Code mnemonics file supplied with the Encyclopedia API software. For workstation platforms the header file name is otc.h.

Output

plCount

The Count is the number of associations for the specified object type.

Return

EAPI_SUCCESSFUL_RC
EAPI_OTC_INVALID_RC

Related Functions

EApiFetchSchemaAscForObjType

Example

```
#include <eapidef.h>
EAPIRC rc = EAPI_SUCCESSFUL_RC;
OBJTYPECODE nObjTypeCode;
LCOUNT lCount;
rc = EApiCountSchemaAscForObjType((OBJTYPECODE) nObjTypeCode,
(PLCOUNT) &lCount);
if ( rc != EAPI_SUCCESSFUL_RC )
    printf("Error in EApiCountSchemaAscForObjType\n");
else
    printf("Count = %ld\n", lCount);
```

Chapter 16: API Return Code Definitions

This chapter describes all the Encyclopedia API return codes and their definitions.

This section contains the following topics:

API Return Codes

The following table lists and defines the API Return Codes:

API Return Code	Return Code Definition
EAPI_ASSOC_NOT_FOUND_RC	association occurrence not found
EAPI_ATC_INVALID_RC	association type code is invalid
EAPI_AUTH_NOT_FOUND_RC	user/group not authorized for model or subset
EAPI_CHECKOUT_NOT_FOUND_RC	checkout not found
EAPI_CONNECT_FAILED_RC	connection not successful
EAPI_DEST_OBJ_NOT_FOUND_RC	destination object not found
EAPI_GROUP_NOT_FOUND_RC	group not found
EAPI_LOCK_NOT_FOUND_RC	lock not found
EAPI_MAXIMUM_COUNT_REACHED_RC	successful and if the number found is greater than the number input. This indicates the function call is successful but that the maximum count has been reached.
EAPI_MNEMONIC_INVALID_RC	mnemonic is invalid
EAPI_MODEL_ALREADY_LOCKED_RC	model already locked
EAPI_MODEL_NOT_CHECKED_OUT_RC	model not checked out
EAPI_MODEL_NOT_FOUND_RC	model not found
EAPI_MODEL_PARENT_NOT_FOUND_RC	model has no parent
EAPI_NOT_CONNECTED_RC	not connected to encyclopedia
EAPI_OBJECT_NOT_FOUND_RC	object not found
EAPI_OBJECT_NOT_IN_SUBSET	object not defined in subset
EAPI_OTC_INVALID_RC	object type code is invalid

API Return Code	Return Code Definition
EAPI_PTC_INVALID_RC	property type code is invalid
EAPI_SOURCE_OBJ_NOT_FOUND_RC	source object not found
EAPI_SUBSET_NOT_CHECKED_OUT_RC	subset not checked out
EAPI_SUBSET_NOT_FOUND_RC	subset not found
EAPI_SUCCESSFUL_RC	the Encyclopedia API Function Call was successful
EAPI_USER_NOT_FOUND_RC	user or group not found
EAPI_USER_NOT_LOGGED_ON_RC	user not logged on
EAPI_MODEL_LOCKED_RC	a model is in use
EAPI_MODEL_INCOMPATIBLE_RC	the model schema is incompatible
EAPI_MODEL_FATAL_ERROR_RC	a fatal error occurred while opening a workstation model
EAPI_MODEL_DIR_READ	the model directory cannot be read when opening a workstation model
EAPI_MODEL_BAD_ERRFILE_RC	the input error file cannot be opened when opening a workstation model
EAPI_INVALID_DIR_PASSWORD_RC	The password given is not the password on file for the given user Id
EAPI_DIRUSER_NOT_FOUND_RC	The userid entered is not present on On the file .
EAPI_PASSWORD_DECRYPTION_FAILED_RC	The decryption of the Password has failed when trying to compare the existing password in file with the password given.
EAPI_DIRUSER_NOT_ACTIVE_RC	The user entered is not an active user
EAPI_SUCCESSFUL_RC	The password given is the password on file for the given user Id
EAPI_FUNCTION_FAULT_RC	Some fault occurred and the validity of the password could not be determined. The fault can be either the clients are Not started or the failure of validation of arguments

Chapter 17: Metamodel Diagrams

All CA Gen models are stored within a defined structure. This defined structure is also a model, called the metamodel. The term meta is used to indicate recursion and self-definition. Therefore, the metamodel is, simply, a model of models. The metamodel is hierarchical in nature.

This section contains the following topics:

CA Gen Models Stored in the Metamodel

This section provides diagrams of the way data is stored in the metamodel. The diagrams should help provide a pictorial view of the data you are trying to retrieve using the Encyclopedia API.

Diagrams represented include:

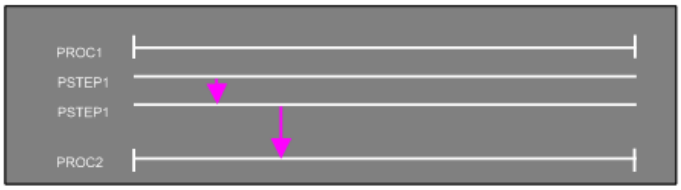
- Dialog Flow
- Data Modeling
- Technical Design Metamodel Definition
- Data Structure List

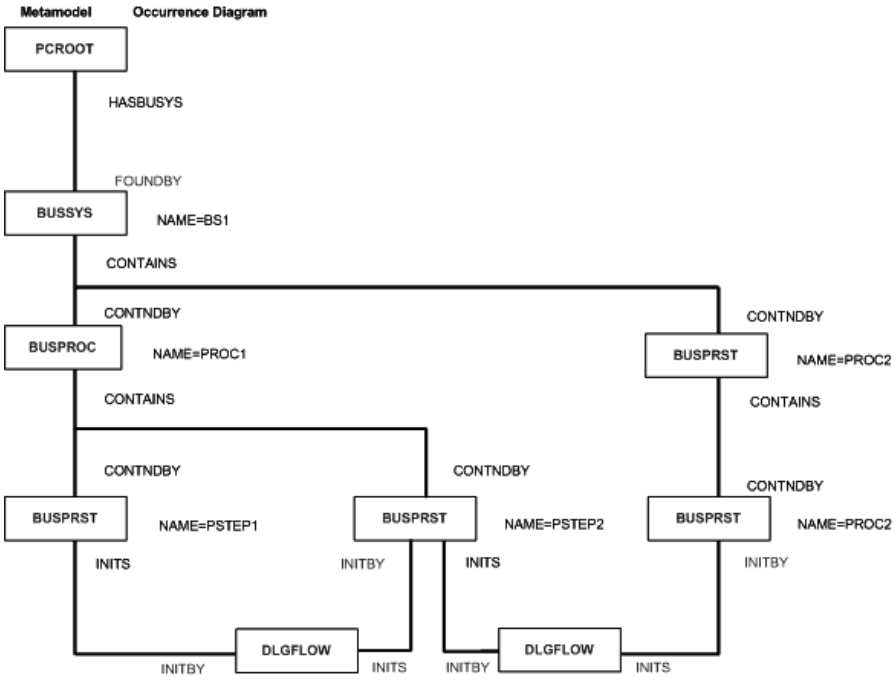
- Metamodel Occurrence
- Activity Hierarchy
- Matrix Processor
- Activity Dependency Metamodel Definition
- Packaging Metamodel Definition

The Data Modeling, Activity Hierarchy, Dialog Flow and Packaging diagrams provide a CA Gen example looking at populated data. The Metamodel Occurrence diagrams provide a view of how each example is represented in the metamodel. The definition diagrams provide a view of how each is displayed. They are definitions of the occurrence diagrams.

Dialog Flow Diagram Objects

CA Gen Diagram

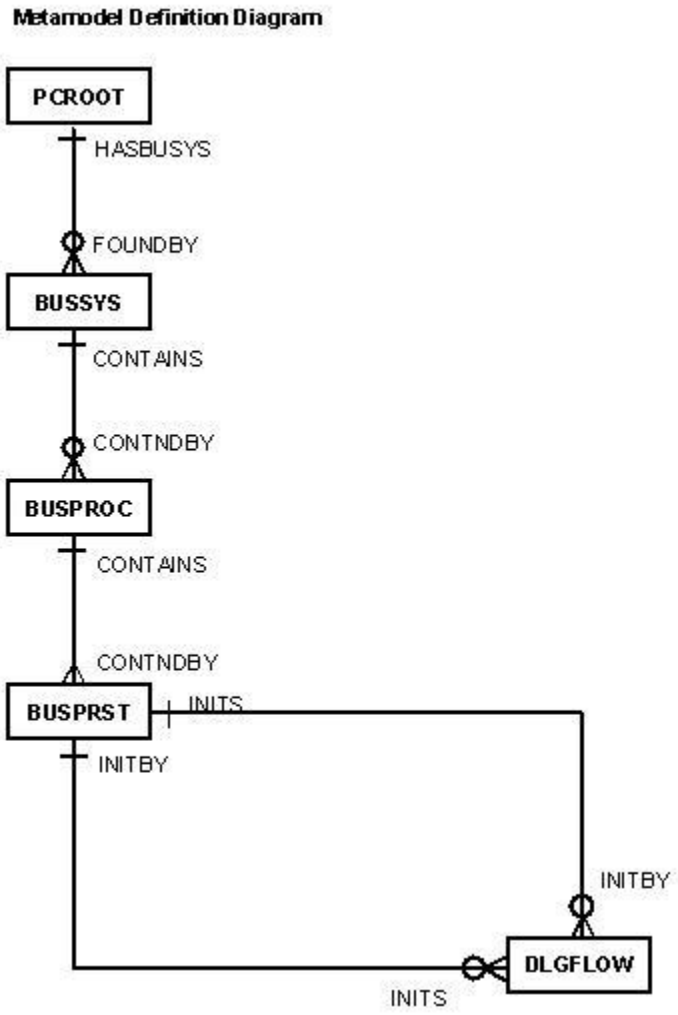




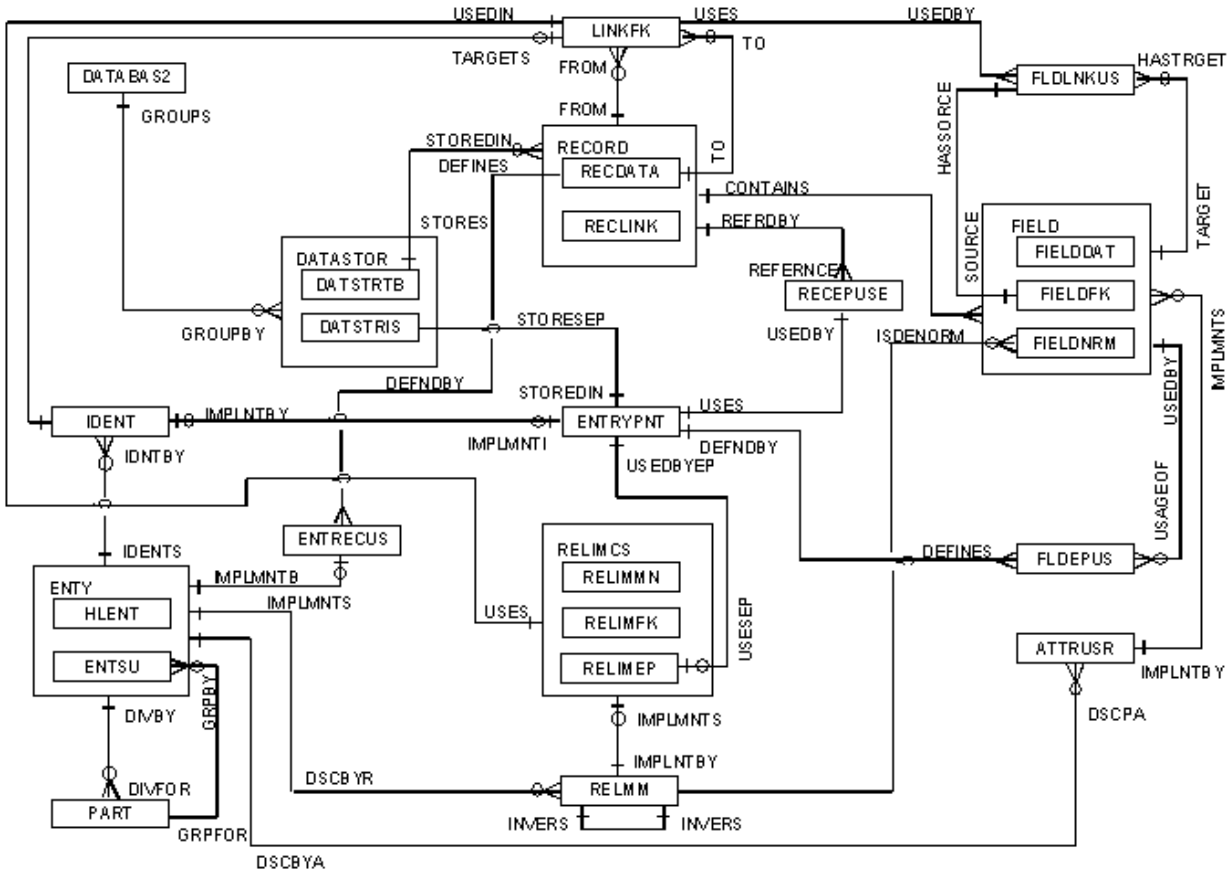
Notes

1. Procedure steps are only displayed if more than one procedure step exists within a procedure.
2. Order of procedures defined by order of BUSSYS CONTAINS association.
3. Order of procedure steps within procedure defined by order of BUSPROC CONTAINS association.
4. Order of dialog flows defined by set of rules.
    * all flows drawn in their own column
    * flows displayed from procedure if only 1 procedure step within procedure
    * flows displayed from procedure step if more than 1 procedure step within procedure
    * flows from procedure step all displayed together
        * flows going up displayed to left of flows going down
        * flows going up displayed longest to shortest
        * flows going down displayed shortest to longest

Dialog Modeling Diagram Objects



Technical Design Metamodel Definition Diagram



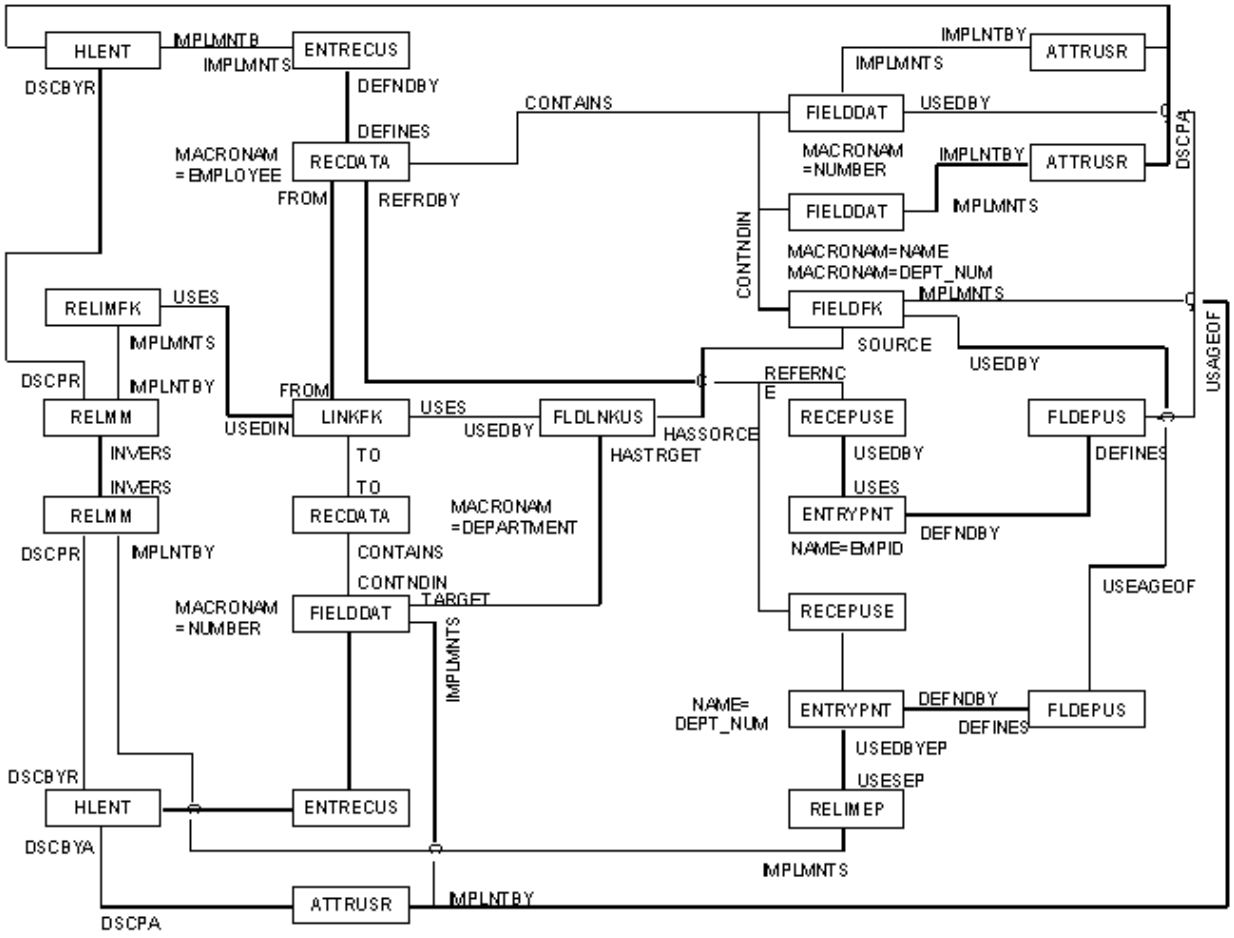
Data Structure List

Type	Macro Name	Format	Size	Optionality	Sequence
Record	DEPARTMENT		32		
Record	EMPLOYEE		4	Not Null	
Field	NUMBER	Integer	25	Not Null	
Field	NAME	Text	3	Null	
FK Field	DEPT_NUM	Small			
Linkage	<No Name> DEPARTMENT		4		
Entry Point	EMPID(PRIMARY)		4	Not Null	Ascend
Field	NUMBER	Integer			
			3		
Entry Point	DEPTNUM	Small	3		
Field	DEPTNUM			Null	Ascend

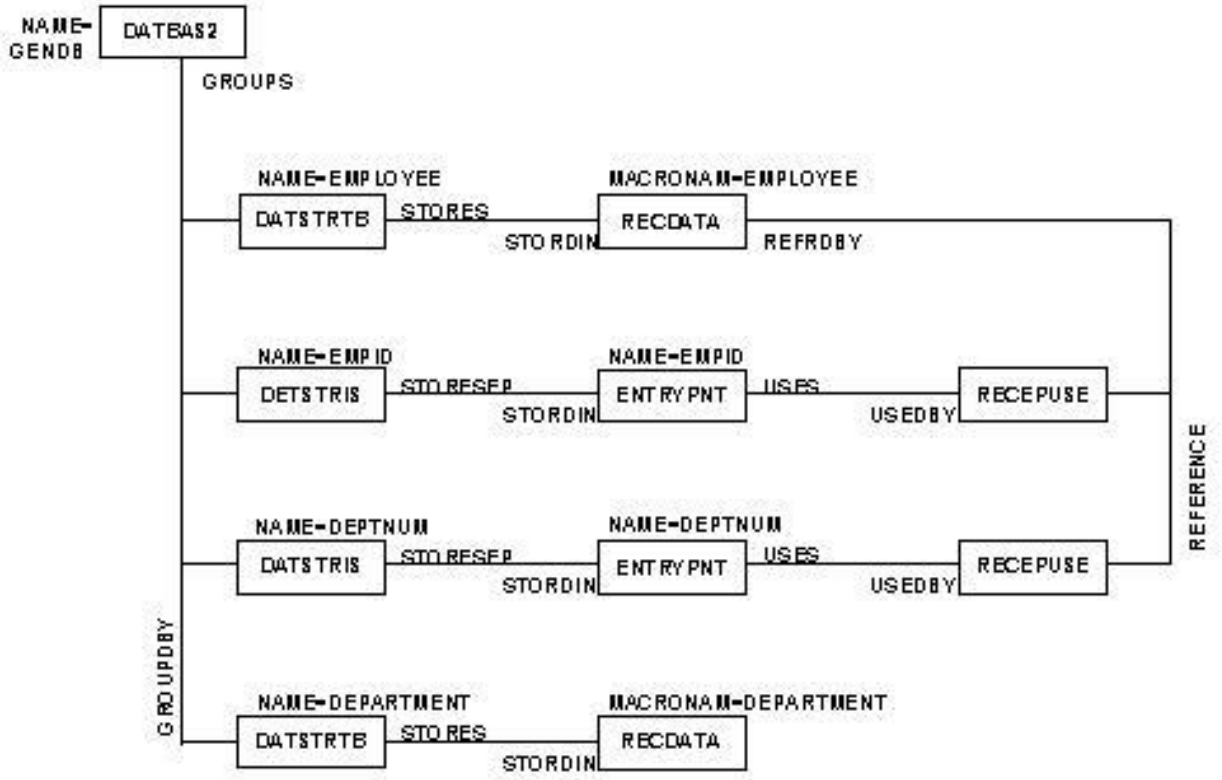
Data Store List
CA Gen Diagram

Type	Name	Quantity	VCAT/STG	LKsz	Subpg	Bufpool
Database	GENDB	roc{55, 34} sum		Page		BPO
TableSpace	EMPLOYEE	roc{8, 4}	IEFVCAT			
Record	EMPLOYEE					BPO
IndexSpace	EMPID	roc{3, 2}	IEFVCAT		16	BPO
IndexSpace	DEPTNUM	roc{3, 2}	IEFVCAT		16	
	DEPARTMENT					
TableSpace	DEPARTMENT	roc{2, 1}	IEFVCAT	Page		BPO
Record						

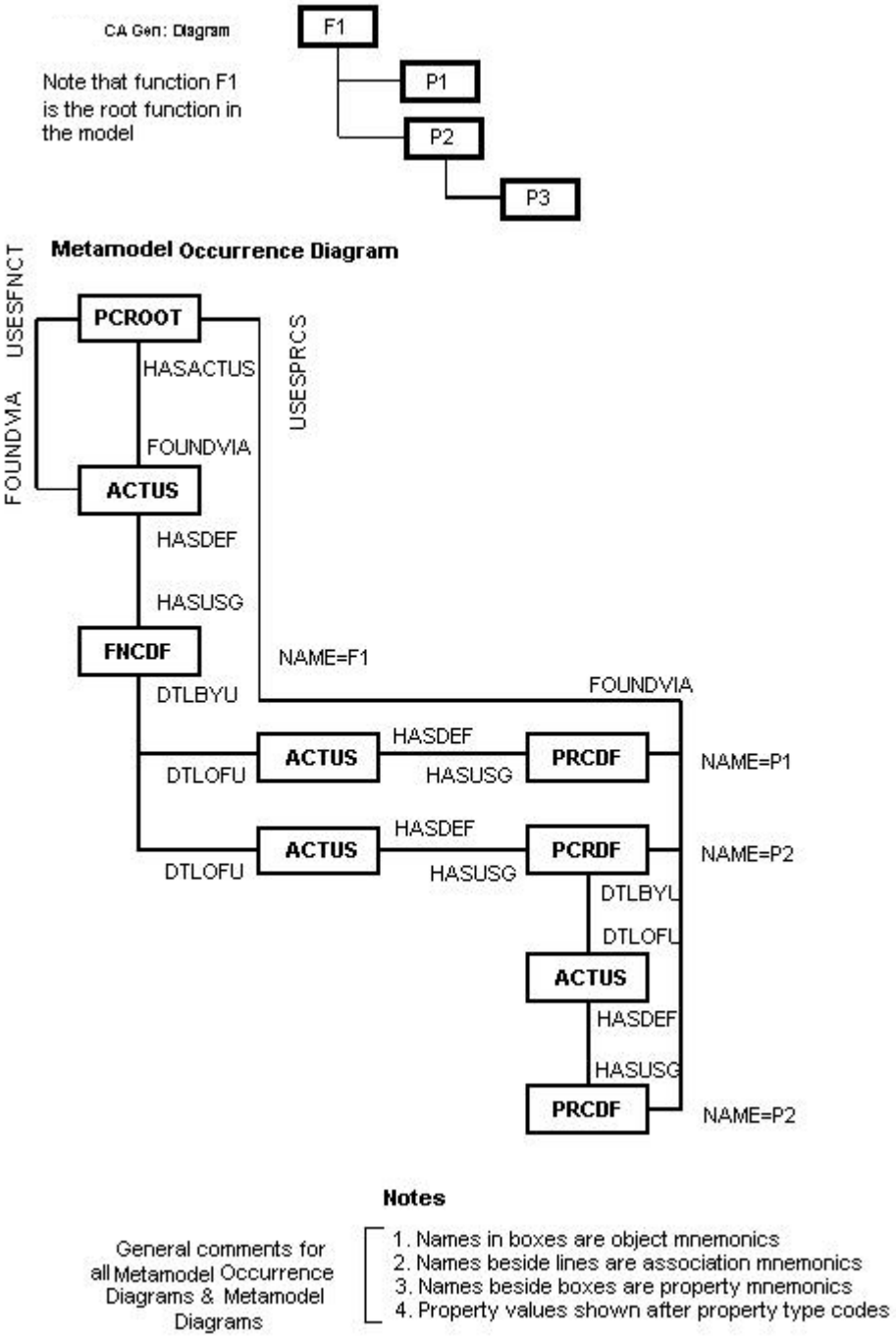
Data Structure List Metamodel Occurrence Diagram



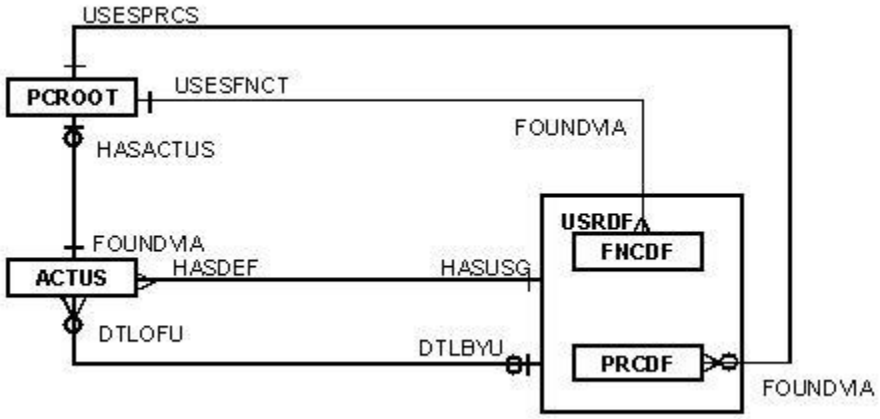
Data Store List Metamodel Occurrence Diagram



Activity Hierarchy Diagram

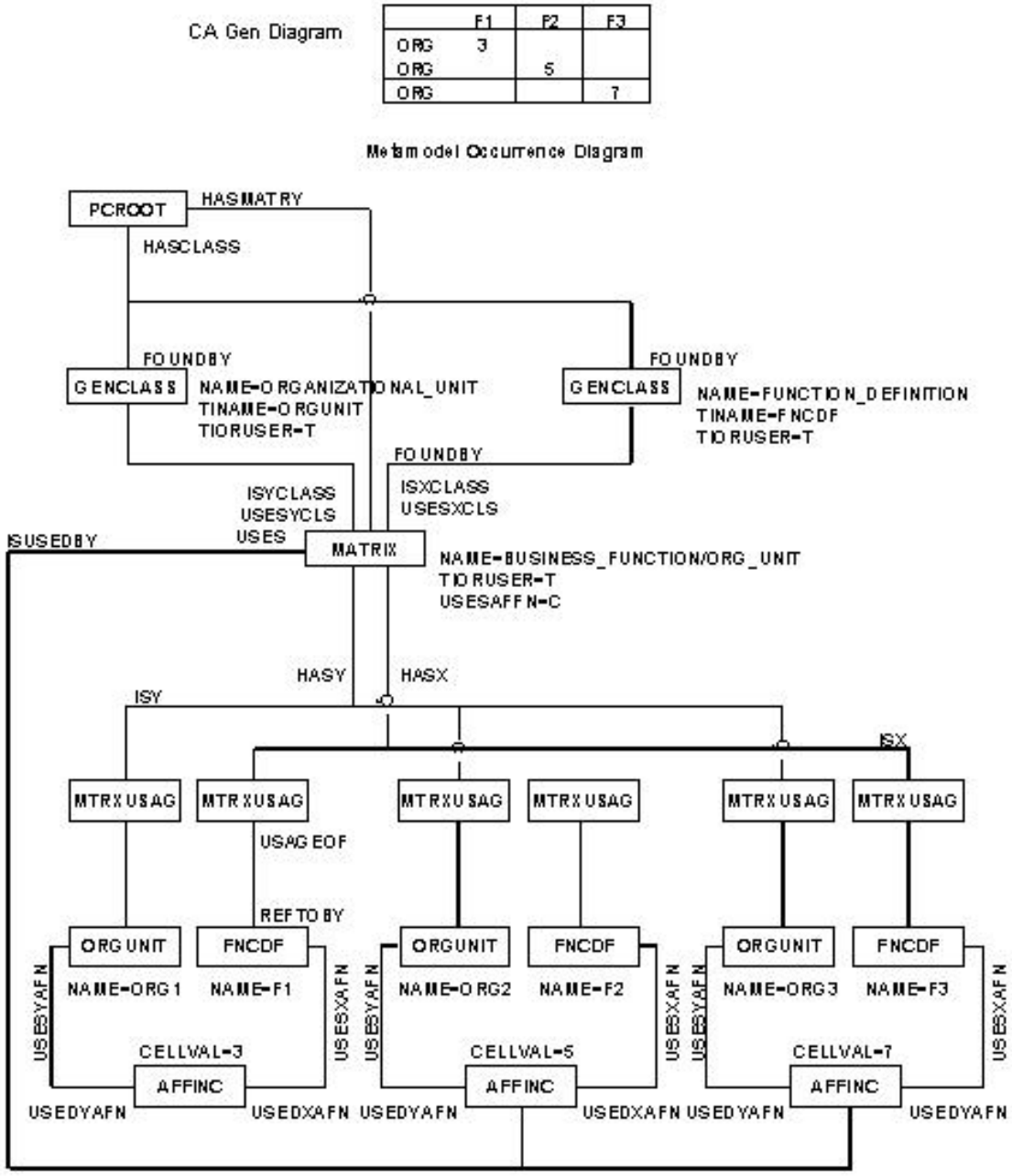


Activity Hierarchy Diagram Objects Metamodel Definition Diagram



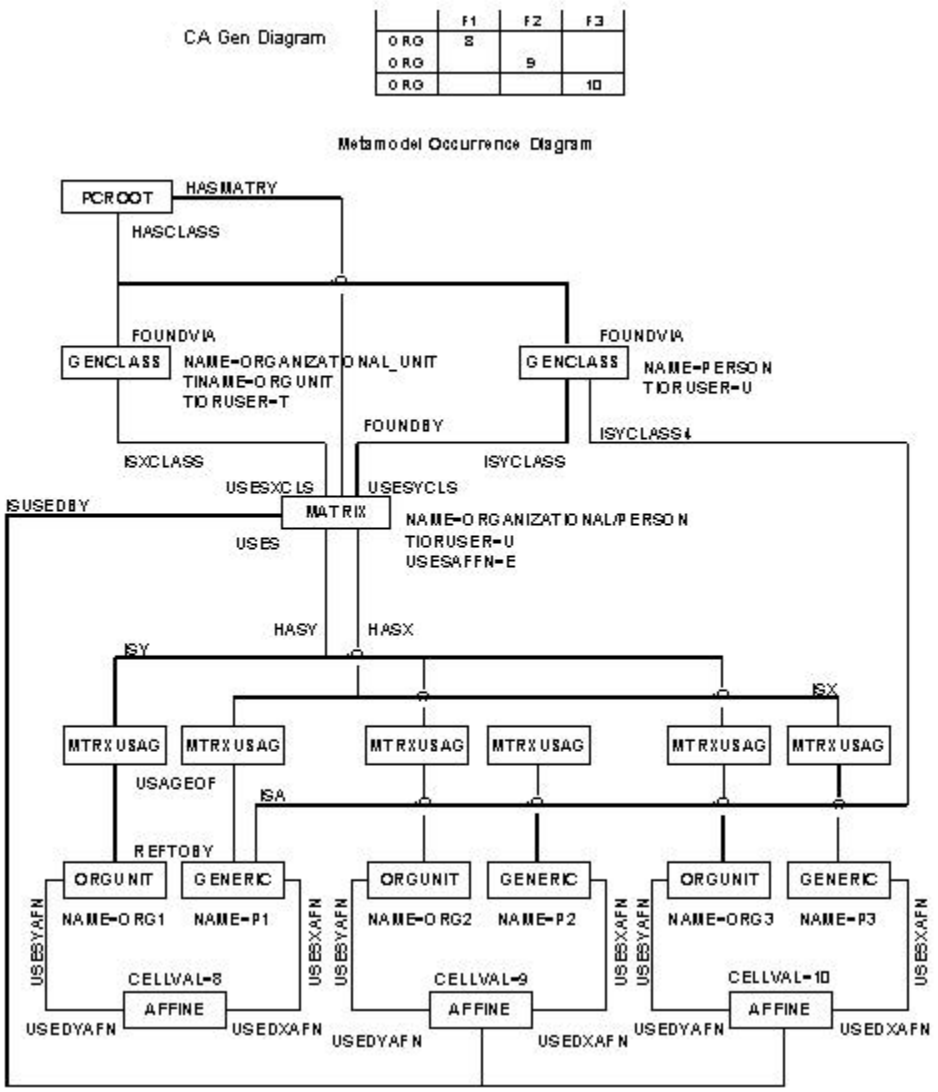
Metamodel Occurrence Diagram Example 1

The first example is an example of the first scenario. It could also be used to see the structure of the second example by changing the TIORUSER property on the matrix object to 'U'.



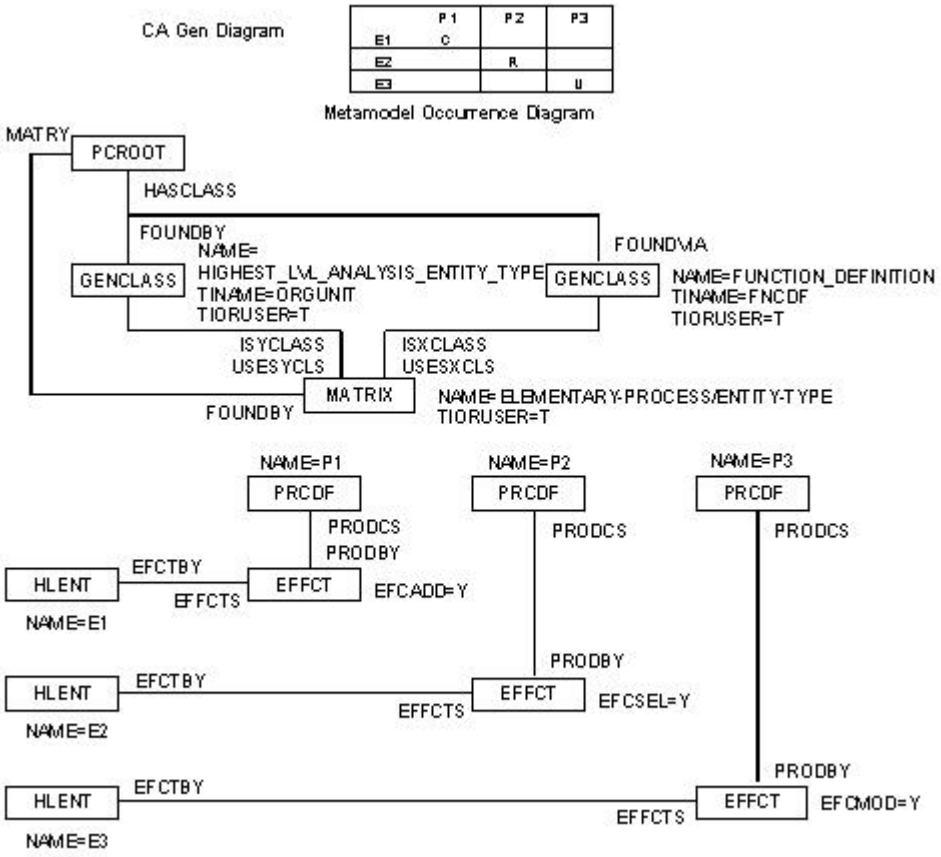
Metamodel Occurrence Diagram Example 2

The second example is an example of the third scenario. Note that in this example only the Y axis has the user defined GENCLASS,GENERIC objects defined.



Metamodel Occurrence Diagram Example 3

The third example is an example of the fourth scenario. In this example, the EFFECT is used as the cell value object.



Metamodel Occurrence Diagram Example 4

There are several types of matrices displayed in the matrix processor. The scenarios are as follows:

- System defined matrices using affinity objects

  These matrices are pre-defined matrices, use pre-defined genclass objects and use the affinity objects to represent cell values.

- User defined matrices / pre-defined genclass objects

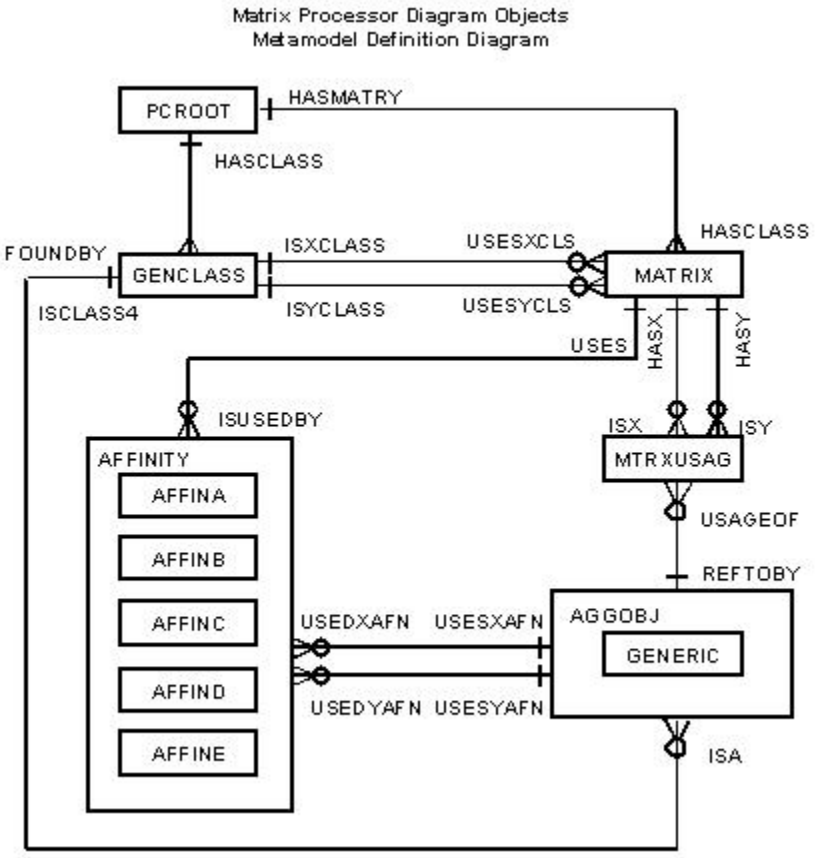
  These matrices are user defined matrices instead of pre-defined matrices but otherwise they are structured the same as listed in the first point.

- User defined matrices and genclass objects

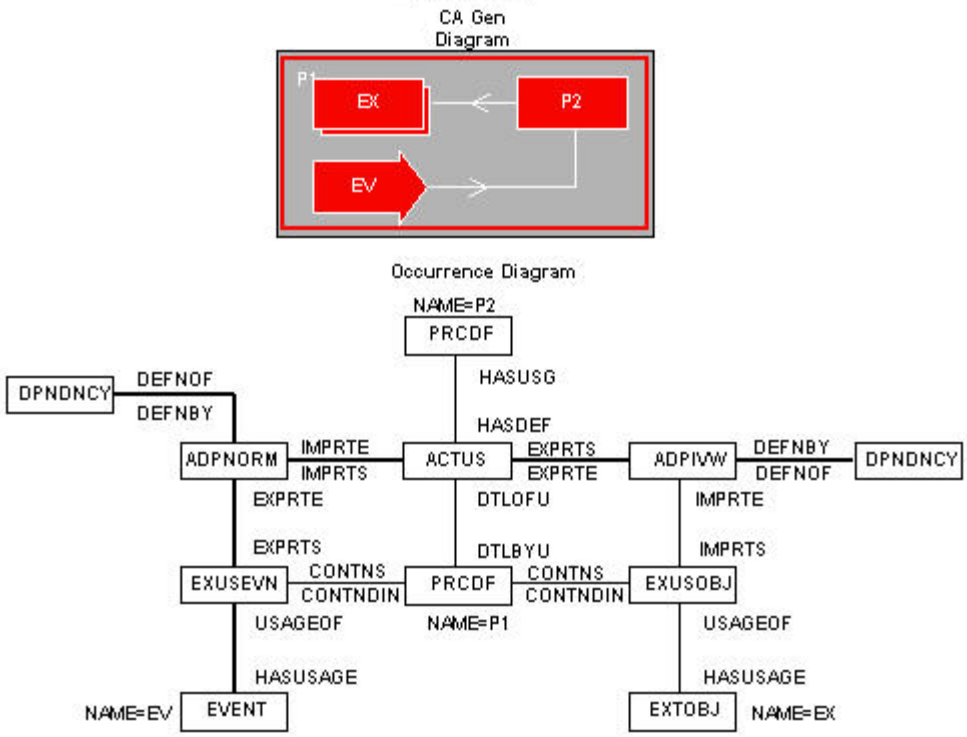
  These matrices are user defined matrices instead of pre-defined matrices and have user defined genclass / generic objects on the x or y axis, or both axis.

- Special cases

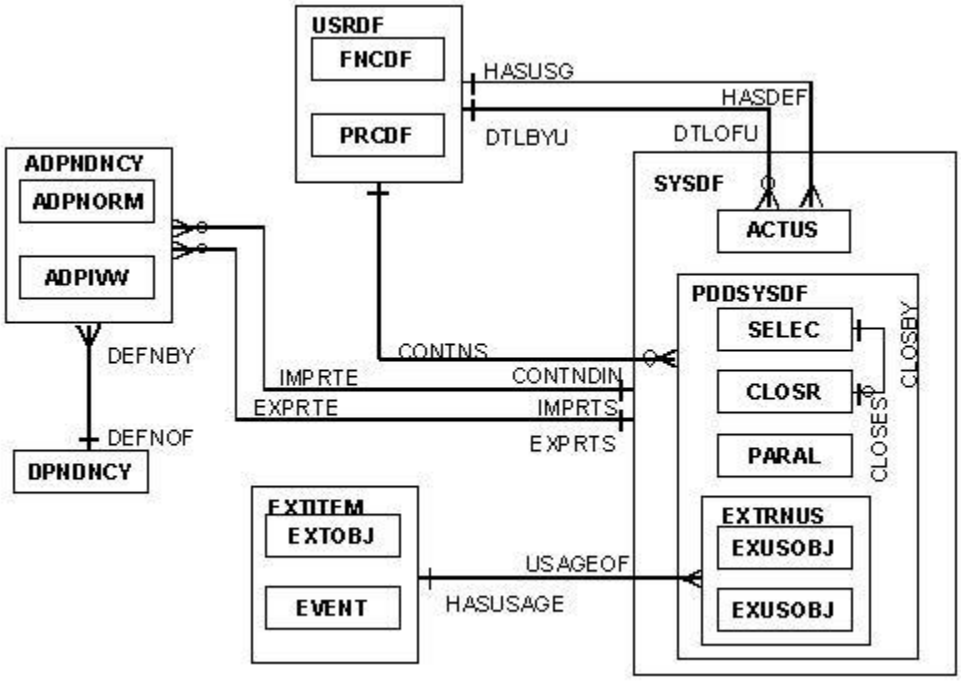
  Matrices like the process / entity type matrix use the expected effects objects to define the cell values instead of the affinity objects.



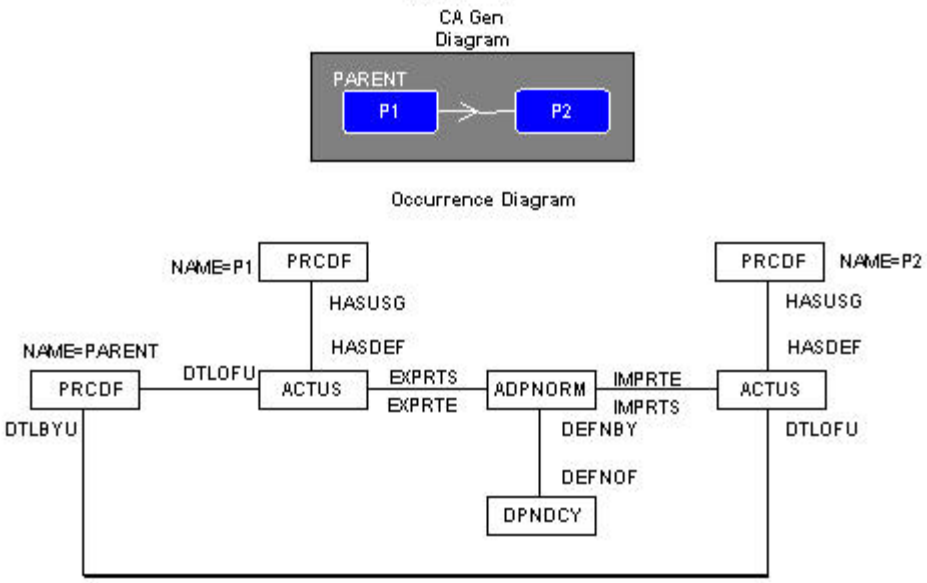
Activity Dependency Diagram



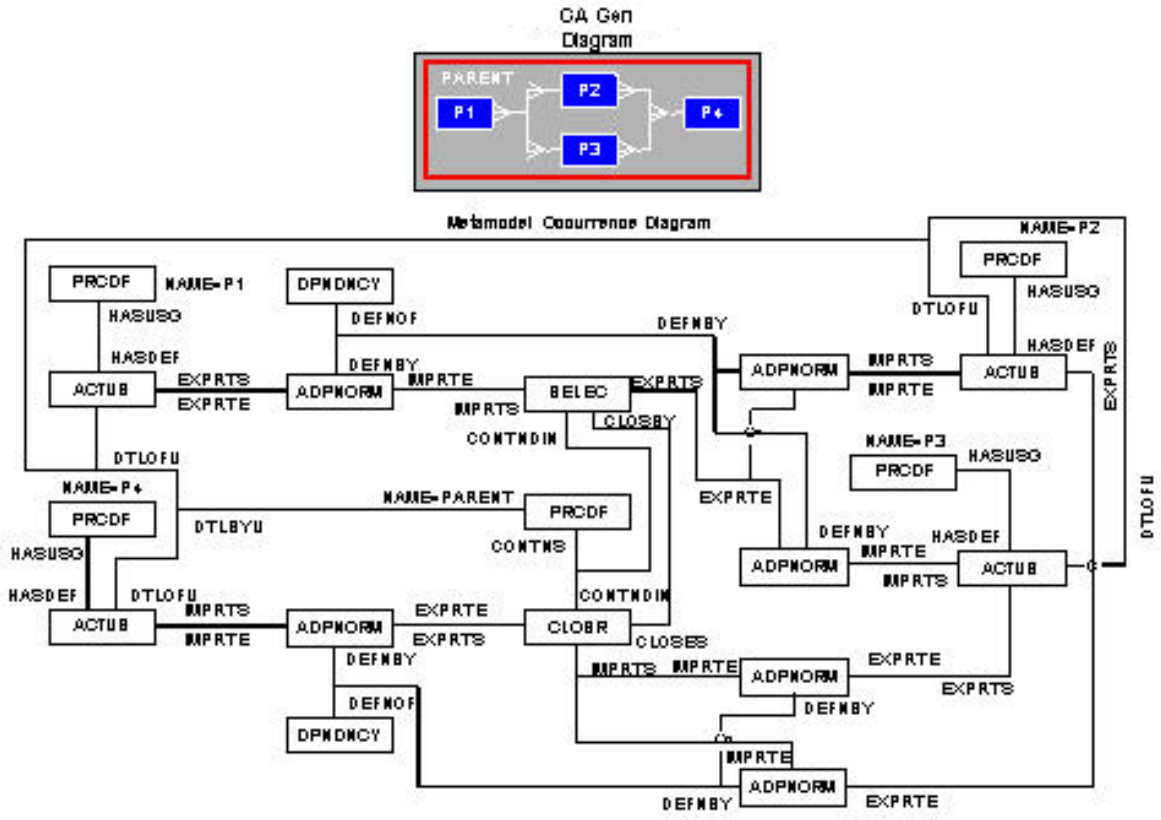
Activity Dependency Diagram Objects Metamodel Definition Diagram



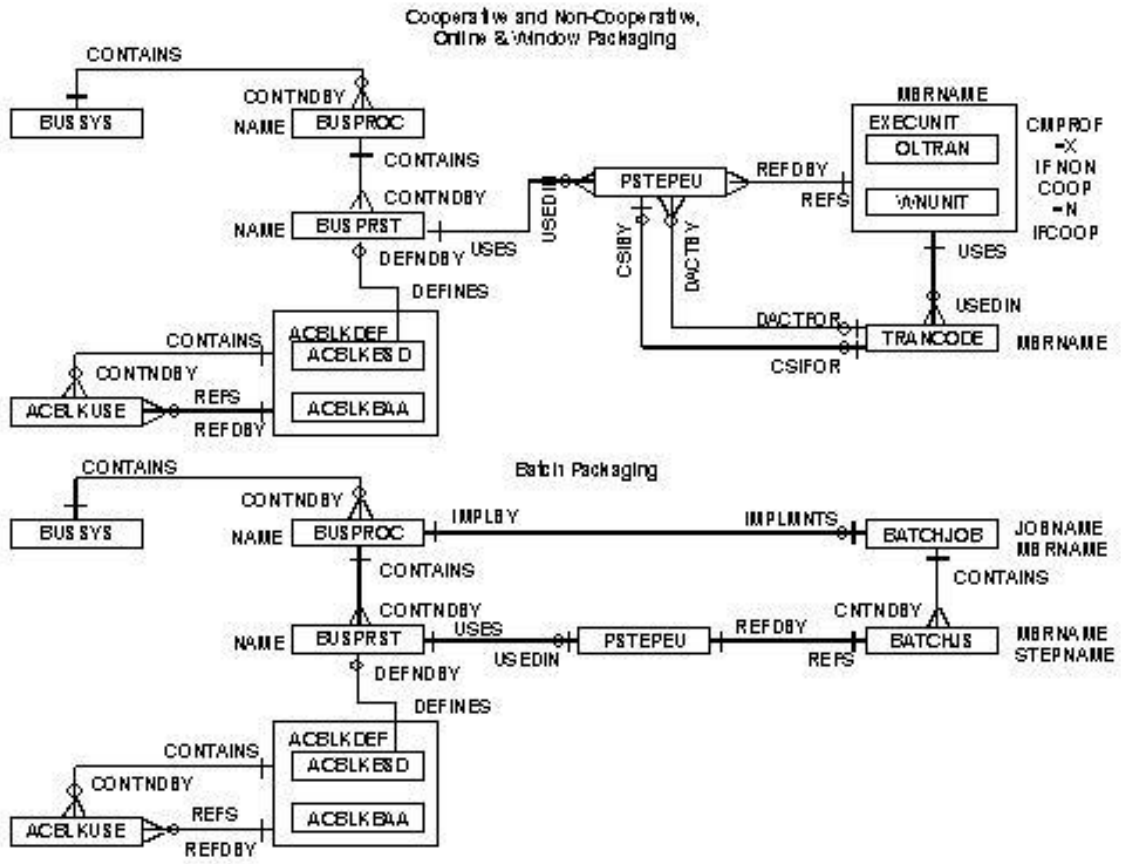
Active Dependency Diagram



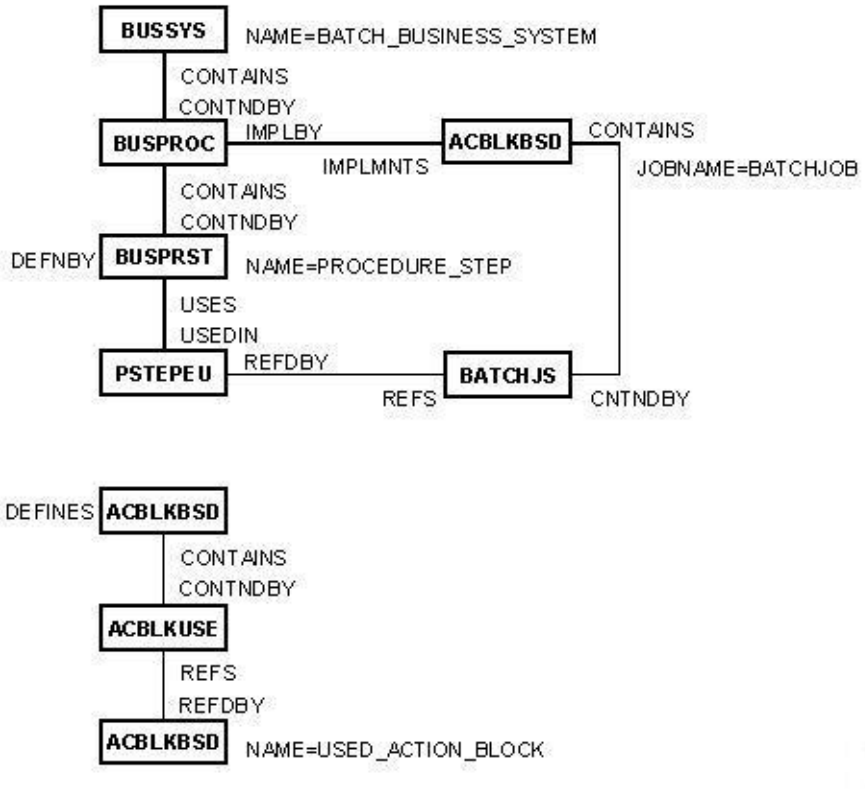
Activity Dependency Diagram Metamodel Occurrence Diagram



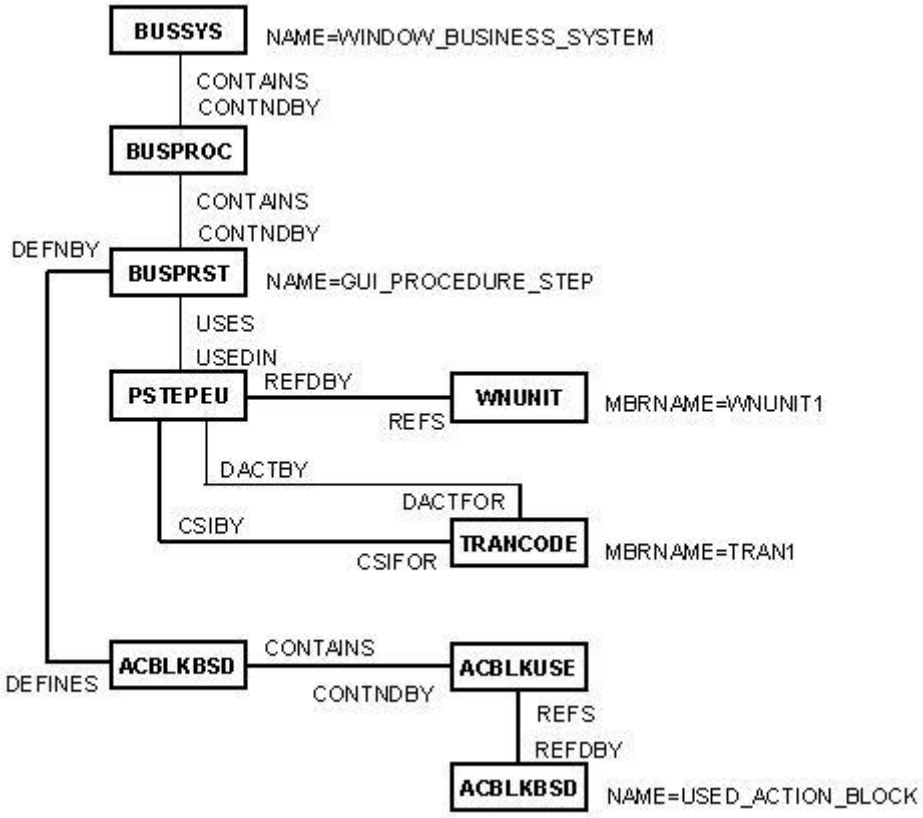
Packaging Metamodel Definition Diagram Objects



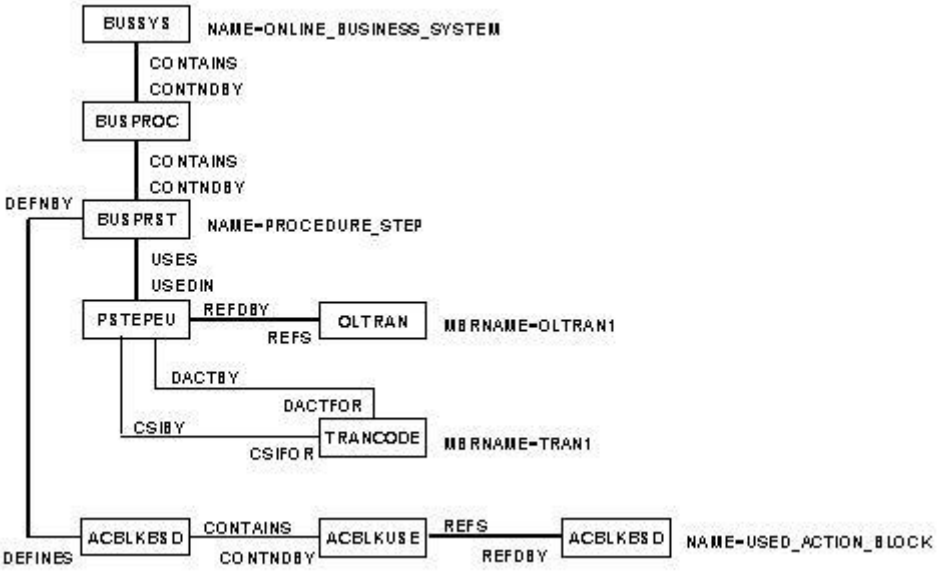
Metamodel Occurrence Diagram - Batch Packaging



Metamodel Occurrence Diagram - Window Packaging



Metamodel Occurrence Diagram - Online Packaging



Packaging

ONLINE_PACKAGING

CA Gen Diagram

Type	Name
Business_Sys	ONLINE_BUSINESS_SYSTEM
Dialog_Manager	OLTRAN1
Proc_Step	PROCEDURE_STEP
Action_Block	USED_ACTION_BLOCK

Window_Packaging

Type	Name
Business_Sys	WINDOW_BUSINESS_SYSTEM
Dialog_Manager	WNUNIT1
Proc_Step	GUI_PROCEDURE_STEP
Action_Block	USED_ACTION_BLOCK

Batch_Packaging

Type	Name
Business_Sys	BATCH_BUSINESS_SYSTEM
Dialog_Manager	BATCHJOB
Proc_Step	BATCH_PROCEDURE_STEP
Action_Block	USED_ACTION_BLOCK

Appendix A: Encyclopedia and Workstation API Functions

This appendix describes the read and Update APIs for the Encyclopedia and Workstation.

List of Read API Functions

This section lists all of the Read API functions. The following table identifies the function and its associated type:

Name of API Function	Type of API Function
EApiConnectToEncy	Encyclopedia
EApiDisconnectEncy	Encyclopedia
EApiCommit	Encyclopedia and Workstation
EApiLogonUserID	Encyclopedia
EApiFetchEncyInfo	Encyclopedia
EApiFetchEncyModelIDs	Encyclopedia
EApiFetchModelInfo	Encyclopedia and Workstation For Workstation API, model_name and code_page are only returned fields. All other fields are defaulted.
EApiFetchModelCreateInfo	Encyclopedia
EApiFetchModelLastUpdateInfo	Encyclopedia
EApiFetchModelParentInfo	Encyclopedia
EApiFetchModelChkoutInfo	Encyclopedia
EApiFetchModelByName	Encyclopedia and Workstation
EApiFetchAllModelObjects	Encyclopedia and Workstation
EApiFetchAllModelTypeObjects	Encyclopedia and Workstation
EApiFetchObjectInfo	Encyclopedia and Workstation
EApiFetchObjectCreateInfo	Encyclopedia
EApiFetchObjectAncestryInfo	Encyclopedia

Name of API Function	Type of API Function
EApiFetchObjectWithAncestryInfo	Encyclopedia
EApiFetchObjectChkoutParentInfo	Encyclopedia and Workstation
EApiFetchSingleCharPrp	Encyclopedia and Workstation
EApiFetchSingleNumericPrp	Encyclopedia and Workstation
EApiFetchCharArrayPrp	Encyclopedia and Workstation
EApiFetchCardOneAsc	Encyclopedia and Workstation
EApiFetchCardOneAscWithObjInfo	Encyclopedia and Workstation
EApiFetchCardManyAsc	Encyclopedia and Workstation
EApiFetchCardManyAscWithObjInfo	Encyclopedia and Workstation
EApiFetchModelNameTypeObjs	Encyclopedia and Workstation
EApiFetchModelObjByNameType	Encyclopedia and Workstation
EApiFetchModelObjListByNameType	Encyclopedia and Workstation
EApiFetchPrpLastUpInfo	Encyclopedia
EApiFetchParentChkoutPrpInfo	Encyclopedia and Workstation
EApiFetchOrderAscOccInfo	Encyclopedia and Workstation
EApiFetchAscOccLastUpdInfo	Encyclopedia
EApiFetchParentAscOccChkoutInfo	Encyclopedia and Workstation
All Subset Functions	Encyclopedia
All Checkout Functions	Encyclopedia
All User/Group Functions	Encyclopedia
All Authorization Functions	Encyclopedia
All Schema Functions	Encyclopedia and Workstation
EApiCountEncyModelIds	Encyclopedia
EApiCountModelObjs	Encyclopedia and Workstation
EApiCountModelTypeObjs	Encyclopedia and Workstation
EApiCountModelNameTypeObjs	Encyclopedia and Workstation
EApiCountSubsetScopingObjs	Encyclopedia
EApiCountChkoutChkdOutObjs	Encyclopedia
EApiCountChkoutsForObj	Encyclopedia
EApiCountCardManyAsc	Encyclopedia and Workstation

Name of API Function	Type of API Function
EApiCountAllUsers	Encyclopedia
EApiCountAllGroups	Encyclopedia
EApiCountUsersInGroup	Encyclopedia
EApiCountGroupsInUser	Encyclopedia
EApiCountUsersGrpsModelAuth	Encyclopedia
EApiCountSchemaAscForObjType	Encyclopedia and Workstation
EApiCountUsersGrpsSubsetAuth	Encyclopedia
EApiCountSchemaPrpForObjType	Encyclopedia and Workstation

List of Update API Functions

This section lists all of the Update API functions. The following table identifies the function and its associated type:

Name of API Function	Type of API Function
EApiAddAssoc	Encyclopedia and Workstation
EApiAddObject	Encyclopedia and Workstation
EApiCommitUpdate	Encyclopedia and Workstation
EApiDelAssoc	Encyclopedia and Workstation
EApiDelObject	Encyclopedia and Workstation
EApiReordrAssocAftr	Encyclopedia and Workstation
EApiReordrAssocBefr	Encyclopedia and Workstation
EApiUpRollback	Encyclopedia and Workstation
EApiUpCharArray	Encyclopedia and Workstation
EApiUpSingleChar	Encyclopedia and Workstation
EApiUpNumeric	Encyclopedia and Workstation

Appendix B: Object Name Validation Rules

This appendix describes the various object name validation rules.

List of Character Set Rules

The character set rules are divided as follows.

Important! The API does not perform validation for string lengths on property updates.

If the name of the object must be unique within its related parent, the name property will not be updated or added until the object is committed. An example of this is, attributes which must be unique within the parent entity. Use the lists of objects and their rules in this appendix to know which objects must have names unique within their related parent.

Non-Construction Object Rule

If the code page of the model is a single byte character, set code page

- Cannot contain the characters: `~!@#$%^&*()+-={}|[]\:";'<>?,./
- Alphabetic characters must be upper case
- Leading spaces are suppressed
- Embedded spaces are replaced with a single underscore

If the code page of the model is a double byte character, set code page cannot contain a DBCS space.

Identifier Rule

- The first character must be alphabetic
- All other characters must be alphabetic, numeric, or _
- Alphabetic characters must be upper case

TD Rule

- The first character must be alphabetic or @#$
- All other characters must be alphabetic, numeric, or @#$_

- The last character cannot be _
- Alphabetic characters must be upper case

No Underscore TD Rule

- The first character must be alphabetic or @#$
- All other characters must be alphabetic, numeric, or @#$
- Alphabetic characters must be upper case

No NLS Rule

- The first character must be alphabetic
- All other characters must be alphabetic or numeric
- Alphabetic character must be upper case

Mixed Case TD Rule

- The first character must be alphabetic or @#$
- All other characters must be alphabetic, numeric, or @#$_
- The last character cannot be _

Oracle TD Rule

- The first character must be alphabetic or #$
- All other characters must be alphabetic, numeric, or #$_
- The last character cannot be _
- Alphabetic characters must be upper case

ODBC TD Rule

- The first character must be alphabetic or @#$_
- All other characters must be alphabetic, numeric, or @#$_
- The last character cannot be _

XML Rule

- The first character must be alphabetic, underscore, or colon
- All other characters must be alphabetic, numeric, hyphen, underscore, colon, or period

Planning Objects

The following table describes the planning objects.

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
ACTIVITY CLUSTER	NAME	Unique within all activity clusters	32	Non-Construction Object Rule
BUSINESS AREA	NAME	Unique within all business areas	32	Non-Construction Object Rule
BUSINESS GOAL	NAME	Unique within all business goals	32	Non-Construction Object Rule
BUSINESS LOCATION	NAME	Unique within all business locations	32	Non-Construction Object Rule
BUSINESS OBJECTIVE	NAME	Unique within all business objectives	32	Non-Construction Object Rule
CRITICAL SUCCESS FACTOR	NAME	Unique within all critical success factors	32	Non-Construction Object Rule
CURRENT DATA BASE OR STORE	NAME	Unique within all current data base or stores	32	Non-Construction Object Rule
CURRENT INFORMATION SYSTEM	NAME	Unique within all current information systems	32	Non-Construction Object Rule
DATA CLUSTER	NAME	Unique within all data clusters	32	Non-Construction Object Rule
FACILITY	NAME	Unique within all facilities	32	Non-Construction Object Rule
HW/SW ENVIRONMENT	NAME	Unique within all hw/sw environments	32	Non-Construction Object Rule
INFORMATION NEED	NAME	Unique within all information needs	32	Non-Construction Object Rule
ORGANIZATIONAL UNIT	NAME	Unique within all organizational units	32	Non-Construction Object Rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
MATRIX	NAME	Unique within all matrices	32	Non-Construction Object Rule

Analysis Objects

The following table describes the analysis objects:

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
ALIAS	NAME	Unique within parent entity type, entity subtype, or attribute	32	Non-Construction Object Rule
ATTRIBUTE	NAME	Unique within all attributes in the entity type's hierarchy	32	Non-Construction Object Rule
ATTRIBUTE	DSDNAME	Unique within all attributes in the entity type's hierarchy	32	Mixed Case Rule
COMMON ACTION BLOCK	NAME	Unique within all functions, processes, procedure steps, and common action blocks	32	Non-Construction Object Rule
DEPENDENCY	NAME	Unique within all dependencies imported by or exported by the functions, processes, external objects, or events that the dependency is imported by or exported by	32	Non-Construction Object Rule
ENTITY SUBTYPE	NAME	Unique within all entity types, entity subtypes, and work attribute sets	32	Non-Construction Object Rule
ENTITY SUBTYPE	DSDNAME	Unique within all entity types and entity subtypes	32	Mixed Case Rule
ENTITY TYPE	NAME	Unique within all entity types, entity subtypes, and work attribute sets	32	Non-Construction Object Rule
ENTITY TYPE	DSDNAME	Unique within all entity types and entity subtypes	32	Mixed Case Rule
EVENT	NAME	Unique within all events	32	Non-Construction Object Rule
EXTERNAL OBJECT	NAME	Unique within all external objects	32	Non-Construction Object Rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
FUNCTION	NAME	Unique within all functions, processes, procedure steps, and common action blocks	32	Non-Construction Object Rule
IDENTIFIER	NAME	Unique within parent entity type or entity subtype	32	Identifier Rule
MUTUALLY EXCLUSIVE	NAME	Unique within parent entity type or entity subtype	32	Non-Construction Object Rule
PROCESS	NAME	Unique within all functions, processes, procedure steps, and common action blocks	32	Non-Construction Object Rule
RELATIONSHIP	NAME	Unique within all relationships in the source entity type's hierarchy and within the same destination entity type's hierarchy	32	Non-Construction Object Rule
SUBJECT AREA	NAME	Unique within all subject areas	32	No NLS Rule
WORK ATTRIBUTE	NAME	Unique within all work attributes in the work attribute set	32	Non-Construction Object Rule
WORK ATTRIBUTE SET	NAME	Unique within all entity types, entity subtypes, and work attribute sets	32	Non-Construction Object Rule

Design Objects

The following table describes the design objects:

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
BUSINESS SYSTEM	NAME	Unique within all business systems	32	Non-Construction Object Rule
COMMAND	NAME	Unique within all commands and command synonyms in the business system	8	Non-Construction Object Rule
COMMAND SYNONYM	NAME	Unique within all commands and command synonyms in the business system	8	Non-Construction Object Rule
CUSTOM PROXY	LABELSTR	First 32 characters are unique within all custom proxies	1024	XML object rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
CUSTOM VIDEO PROPERTY	CGNAME	Unique within all custom video properties in the business system	32	Non-Construction Object Rule
DEFAULT EDIT PATTERN	NAME	Unique within all default edit patterns in the business system	32	Non-Construction Object Rule
DIALECT	NAME	Unique within all dialects	8	No NLS Rule
DIALOG BOX	NAME	Unique within all default and non-default dialect primary windows and dialog boxes in the procedure step	32	Non-Construction Object Rule
EXIT STATE	NAME	Unique within all exit states	32	Non-Construction Object Rule
PRIMARY WINDOW	NAME	Unique within all default and non-default dialect primary windows and dialog boxes in the procedure step	32	Non-Construction Object Rule
PROCEDURE	NAME	Unique within all procedures	32	Non-Construction Object Rule
PROCEDURE STEP	NAME	Unique within all functions, processes, procedure steps, and common action blocks	32	Non-Construction Object Rule
TEMPLATE	NAME	Unique within all templates	32	Non-Construction Object Rule
WEB SERVICE DEFINITION	LABELSTR	First 32 characters are unique within all web service definitions	1024	XML object rule
WINDOW PRESENTATION	NAME	Unique within all window presentations	32	Non-Construction Object Rule

Technical Design Objects

The following table describes the technical design objects:

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
TECHNICAL DESIGN DEFAULT	RITRIGNM	Unique within all online load modules, window load modules, batch job steps, implementation logics, external implementation logics, and implementation screens	8	No NLS Rule
DATABASE	NAME	Unique within all base database objects and unique within all extended database objects types that the base database does not extend to.	8	No Underscore TD Rule
DATABASE DB2 z/OS EXTENSION	NAME	Unique within all database DB2 z/OS extension objects and unique within all base database objects that do not extend to a database DB2 z/OS extension object	8	No Underscore TD Rule
DATABASE DB2 UDB EXTENSION	NAME	Unique within all database DB2 UDB extension objects and unique within all base database objects that do not extend to a database DB2 UDB extension object	8	TD Rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
DATABASE MS SQL SERVER EXTENSION	NAME	Unique within all database MS SQL Server extension objects and unique within all base database objects that do not extend to a database MS SQL Server extension object	30	Mixed Case TD Rule
DATABASE ORACLE EXTENSION	NAME	Unique within all database Oracle extension objects and unique within all base database objects that do not extend to a database Oracle extension object	8	Oracle TD Rule
DATABASE ODBC EXTENSION	NAME	Unique within all database ODBC extension objects and unique within all base database objects that do not extend to a database ODBC extension object	32	ODBC TD Rule
DATABASE NonStop SQL/MP EXTENSION	NAME	Unique within all database NonStop SQL/MP extension objects and unique within all base database objects that do not extend to a database NonStop SQL/MP extension object	8	Identifier Rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
DATA/ DENORMALIZED / FOREIGN KEY COLUMN	MACRONAM	Unique within all base data/denormalized/foreign key columns in the table and unique within all extended data/denormalized/foreign key columns in the table that the base data/denormalized/foreign key column does not extend to	30	TD Rule
DATA/ DENORMALIZED / FOREIGN KEY COLUMN DB2 z/OS EXTENSION	NAME	Unique within all data/denormalized/foreign key column DB2 z/OS extension objects in the table and unique within all names of base data/denormalized/ foreign key columns that do not extend to a data/ denormalized/foreign key DB2 z/OS extension object	30	TD Rule
DATA/ DENORMALIZED / FOREIGN KEY COLUMN DB2 UDB EXTENSION	NAME	Unique within all data/denormalized/foreign key column DB2 UDB extension objects in the table and unique within all names of base data/ denormalized/foreign key columns that do not extend to a data/denormalized/foreign key DB2 UDB extension object	30	TD Rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
DATA/ DENORMALIZED / FOREIGN KEY COLUMN MS SQL SERVER EXTENSION	NAME	Unique within all data/denormalized/foreign key column MS SQL Server extension objects in the table and unique within all names of base data/ denormalized/foreign key columns that do not extend to a data/denormalized/foreign key MS SQL Server extension object	30	Mixed Case TD Rule
DATA/ DENORMALIZED / FOREIGN KEY COLUMN ORACLE EXTENSION	NAME	Unique within all data/denormalized/foreign key column Oracle extension objects in the table and unique within all names of base data/denormalized/ foreign key columns that do not extend to a data/ denormalized/foreign key Oracle extension object	30	Oracle TD Rule
DATA/ DENORMALIZED / FOREIGN KEY COLUMN ODBC EXTENSION	NAME	Unique within all data/denormalized/foreign key column ODBC extension objects in the table and unique within all names of base data/denormalized/ foreign key columns that do not extend to a data/ denormalized/foreign key ODBC extension object	32	ODBC TD Rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
DATA/ DENORMALIZED / FOREIGN KEY COLUMN JDBC EXTENSION	NAME	Unique within all data/denormalized/foreign key column JDBC extension objects in the table and unique within all names of base data/denormalized/ foreign key columns that do not extend to a data/ denormalized/foreign key JDBC extension object	32	ODBC TD Rule
DATA/ DENORMALIZED / FOREIGN KEY COLUMN NonStop SQL/MP EXTENSION	NAME	Unique within all data/denormalized/foreign key column NonStop SQL/MP extension objects in the table and unique within all names of base data/denormalized/ foreign key columns that do not extend to a data/ denormalized/foreign key NonStop SQL/MP extension object	30	Identifier Rule
DATA/LINK TABLE	MACRONAM	Unique within all base data/link tables and unique within all names of extended data/link tables that the base data/link table does not extend to	32	TD Rule
DATA/LINK TABLE DB2 z/OS EXTENSION	NAME	Unique within all data/link table DB2 z/OS extension objects and unique within all names of base data/link tables that do not extend to a data/link table DB2 z/OS extension object	32	TD Rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
DATA/LINK TABLE DB2 UDB EXTENSION	NAME	Unique within all data/link table DB2 UDB extension objects and unique within all names of base data/link tables that do not extend to a data/link table DB2 UDB extension object	32	TD Rule
DATA/LINK TABLE MS SQL SERVER EXTENSION	NAME	Unique within all data/link table MS SQL Server extension objects and unique within all names of base data/link tables that do not extend to a data/link table MS SQL Server extension object	30	Mixed Case TD Rule
DATA/LINK TABLE MS SQL SERVER EXTENSION	RINAME	Unique within all data/link table MS SQL Server extension objects	26	Mixed Case TD Rule
DATA/LINK TABLE ORACLE EXTENSION	NAME	Unique within all data/link table Oracle extension objects and unique within all names of base data/link tables that do not extend to a data/link table Oracle extension object	30	Oracle TD Rule
DATA/LINK TABLE ODBC EXTENSION	NAME	Unique within all data/link table ODBC extension objects and unique within all names of base data/link tables that do not extend to a data/link table ODBC extension object	32	ODBC TD Rule
DATA/LINK TABLE ORACLE EXTENSION	RINAME	Unique within all data/link table Oracle extension objects	30	Oracle TD Rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
DATA/LINK TABLE JDBC EXTENSION	NAME	Unique within all data/link table JDBC extension objects and unique within all names of base data/link tables that do not extend to a data/link table JDBC extension object	32	ODBC TD Rule
DATA/LINK TABLE NonStop SQL/MP EXTENSION	NAME	Unique within all data/link table NonStop SQL/MP extension objects and unique within all names of base data/link tables that do not extend to a data/link table NonStop SQL/MP extension object	23	Identifier Rule
ENTRY POINT	NAME	1) Unique within all base entry points in the database and 2) unique within all base tablespaces in the database and 3) unique within all extended entry point types in the database that the base entry point does not extend to and 4) unique within all equivalent extended tablespace types in the database that the base entry point does not extend to	32	No Underscore TD Rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
ENTRY POINT DB2 z/OS EXTENSION	NAME	1) Unique within all entry point DB2 z/OS extension objects in the database and 2) unique within all tablespace DB2 z/OS extension objects in the database and 3) unique within all base entry point objects in the database that do not extend to an entry point DB2 z/OS extension object 4) unique within all base tablespace objects in the database that do not extend to a tablespace DB2 z/OS extension object	32	No Underscore TD Rule
ENTRY POINT DB2 UDB EXTENSION	NAME	1) Unique within all entry point DB2 UDB extension objects in the database and 2) unique within all tablespace DB2 UDB extension objects in the database and 3) unique within all base entry point objects in the database that do not extend to an entry point DB2 UDB extension object 4) unique within all base tablespace objects in the database that do not extend to a tablespace DB2 UDB extension object	18	TD Rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
ENTRY POINT MS SQL SERVER EXTENSION	NAME	1) Unique within all entry point MS SQL Server extension objects in the database and 2) unique within all base entry point objects in the database that do not extend to an entry point MS SQL Server extension object	30	Mixed Case TD Rule
ENTRY POINT ORACLE EXTENSION	NAME	1) Unique within all entry point Oracle extension objects in the database and 2) unique within all tablespace Oracle extension objects in the database and 3) unique within all base entry point objects in the database that do not extend to an entry point Oracle extension object 4) unique within all base tablespace objects in the database that do not extend to a tablespace Oracle extension object	32	Oracle TD Rule
ENTRY POINT ODBC EXTENSION	NAME	1) Unique within all entry point ODBC extension objects in the database and 2) unique within all base entry point objects in the database that do not extend to an entry point ODBC extension object	32	ODBC TD Rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
ENTRY POINT JDBC EXTENSION	NAME	1) Unique within all entry point JDBC extension objects in the database and 2) unique within all base entry point objects in the database that do not extend to an entry point JDBC extension object	32	ODBC TD Rule
ENTRY POINT NonStop SQL/MP EXTENSION	NAME	1) Unique within all entry point NonStop SQL/MP extension objects in the database and 2) unique within all base entry point objects in the database that do not extend to an entry point NonStop SQL/MP extension object	23	Identifier Rule
STORAGE GROUP	NAME	Unique within all storage groups	8	No Underscore TD Rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set Rule
TABLESPACE	NAME	1) Unique within all base tablespaces in the database and 2) unique within all base entry points in the database and 3) unique within all extended tablespace types in the database that the base tablespace does not extend to and 4) unique within all equivalent extended entry point types in the database that the base tablespace does not extend to	8	No Underscore TD Rule

Construction Objects

The following table describes the construction objects:

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set
BATCH JOB	JOBNAME	Unique within all batch jobs	8	No NLS Rule
BATCH JOB STEP	MBRNAME	Unique within all online load modules, window load modules, batch job steps, implementation logics, external implementation logics, implementation screens, operations libraries, z/OS libraries, and the technical design RITRIGNM property	8	No NLS Rule
BATCH JOB STEP	STEPNAME	Unique within all batch job steps	8	No NLS Rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set
CONFIGURATION INSTANCE	LABELSTR	Unique within all configuration instances	32	XML object rule
EXTERNAL IMPLEMENTATION LOGIC	MBRNAME	Unique within all online load modules, window load modules, batch job steps, implementation logics, external implementation logics, implementation screens, operations libraries, z/OS libraries, and the technical design RITRIGNM property	8	No NLS Rule
IMPLEMENTATION LOGIC	MBRNAME	Unique within all online load modules, window load modules, batch job steps, implementation logics, external implementation logics, implementation screens, operations libraries, z/OS libraries, and the technical design RITRIGNM property	8	No NLS Rule
IMPLEMENTATION SCREEN	MBRNAME	Unique within all online load modules, window load modules, batch job steps, implementation logics, external implementation logics, implementation screens, operations libraries, z/OS libraries, and the technical design RITRIGNM property	8	No NLS Rule
IMPLEMENTATION SCREEN	MIDNAME	Unique within all implementation screen midnames, modnames, and fmtnames	8	No NLS Rule
IMPLEMENTATION SCREEN	MODNAME	Unique within all implementation screen midnames, modnames, and fmtnames	8	No NLS Rule
IMPLEMENTATION SCREEN	FMTNAME	Unique within all implementation screen midnames, modnames, and fmtnames	8	No NLS Rule

Object Type	Property Type	Name Uniqueness Rule	Length	Character Set
ONLINE LOAD MODULE	MBRNAME	Unique within all online load modules, window load modules, batch job steps, implementation logics, external implementation logics, implementation screens, operations libraries, z/OS libraries, and the technical design RITRIGNM property	8	No NLS Rule
OPERATIONS LIBRARY	MBRNAME	Unique within all online load modules, window load modules, operations libraries, z/OS libraries, batch job steps, implementation logics, external implementation logics, implementation screens and the technical design RITRIGNM property	8	No NLS Rule
TRANSACTION CODE	MBRNAME	Unique within all trancodes	8	No NLS Rule
WINDOW LOAD MODULE	MBRNAME	Unique within all online load modules, window load modules, batch job steps, implementation logics, external implementation logics, implementation screens, operations libraries, z/OS libraries, and the technical design RITRIGNM property	8	No NLS Rule
z/OS LIBRARY	MBRNAME	Unique within all online load modules, window load modules, operations libraries, z/OS libraries, batch job steps, implementation logics, external implementation logics, implementation screens and the technical design RITRIGNM property	8	No NLS Rule

Appendix C: CA Gen Toolset Automation

This appendix describes the various plug-ins and objects for CA Gen Toolset automation.

Plug-ins

CA Gen allows applications to access the currently opened model through a mechanism known as a "plug-in". This concept lets an application register itself so that the CA Gen toolset knows where it is and how to call it. A menu item will be added to the CA Gen "Plug-in" menu so that the application may be started directly from the toolset. Parameters are passed to the plug-in application indicating what diagram started the application, the main object in the diagram, and any objects that may be selected.

Plug-in Installation

When a plug-in is installed, it will update the appropriate values in the Windows registry to indicate to the CA Gen toolset that the plug-in is available. Each plug-in application will add a folder into the Windows registry under "HKEY_LOCAL_MACHINE\SOFTWARE\ComputerAssociates\CA Gen\xx\Plug-ins" key. For each key, the workstation creates a menu item under the "Plug-in" menu using the text found in the "MenuItemText" value. On a Windows 7 64-bit system, the registry entry is under "HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ComputerAssociates\CA Gen\xx\Plug-ins".

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Each of the new plug-in menu items may be enabled or disabled using a combination of the registry key "EnableInDiagrams", the registry value "EnableSelectedObjectCount", and the registry key "EnableSelectedObjectTypes":

- If there are any diagram keys specified under the "EnableInDiagrams" key, then the menu item will only be enabled in the specified diagrams. If no diagram keys are specified under the "EnableInDiagrams" key, then the menu item may be enabled always, depending on whether or not the "EnableSelectedObjectCount" value is specified and (or) if any object type keys are specified under the "EnableSelectedObjectTypes" key.
- If the "EnableSelectedObjectCount" value is specified, then the menu item will only be enabled when the specified number of objects are selected. If the value is not specified, then the menu item may be enabled regardless of whether or not any objects are selected, depending on whether or not the "EnableSelectedObjectCount" value is specified and (or) if any object type keys are specified under the "EnableSelectedObjectTypes" key.

- If there are any object type keys specified under the "EnableSelectedObjectTypes" key, then the menu item will only be enabled in the given diagram when all the constraints for each object type are satisfied.

For example, the following registry values might be supplied by a registered plug-in application:

- MenuItemText = Switch Business System
- EnableInDiagram = DialogDesign
- EnableSelectedObjectCount = +

This set of registry values implies that the menu item "Switch Business System" will be added under the "Plug-in " menu. This menu item will only be enabled when the Dialog Design diagram is open and one or more objects are selected.

Toolset Initialization

Whenever the "Plug-in " menu is selected, it will query the Windows registry for entries indicating that there are plug-ins available. For each plug-in entry in the registry, a sub-menu item will be added to the toolset menu hierarchy Plug-in.

Plug-in Menu Item Selection

When a plug-in menu item is selected, the CA Gen toolset will execute the associated plug-in application passing the diagram name, the scoping object, and any selected object ids that were specified by the plug-in Windows registry entry. The plug-in menu items will use one of 20 predefined resource ids to indicate which menu item was selected. This implies that the toolset will only be able to display 20 plug-in application menu items at a time. If there are more than 20 applications installed, only the first 20 (as sorted in the registry) will be displayed.

Plug-in Application Execution

The plug-in application will instantiate a pointer to the currently opened CA Gen toolset using the toolset OLE automation interface. This gives the application access to the functionality provided by the interface. In particular, it will allow the application to read or modify the currently opened model using the meta-model API functions added to the toolset OLE automation interface by this project as well as other functions already provided by the interface.

The meta-model API functions are provided in three objects:

- **OpenAPI** - This object provides non-object specific functionality such as converting a mnemonic value to a number, counting the number of objects in the model, or returning a **MetaModelObjects** collection object pointing to a list of all objects of a specified type in the model. It is returned when the **OpenAPI** property of the **Model** object is selected.
- **MetaModelObjects** - This collection object is returned by any function that might return more than one **MetaModelObject**. It provides a single "count" property and a single "Item" method.
- **MetaModelObjec**t - This object provides object specific functionality such as fetch or update property or add or follow an association. If the object_id is known, the object may be fetched by using the **OpenAPI** functi*on GetMetaModelObj*ect or it may be retrieved as an item of a **MetaModelObjects** collection.

Plug-in Application Registration

To register an external application as a CA Gen toolset plug-in, set the registry keys at HKEY_LOCAL_MACHINE\Software\ComputerAssociates\CA Gen\Plug-ins on Windows 32-bit computer and HKEY_LOCAL_MACHINE\Software\Wow6432Node\ComputerAssociates\CA Gen\Plug-ins on a Windows 64-bit computer.

The following items will be allowed in the registry:

- Plug-in Application name (required key) - This will be a key that the registry values will be added to. The name will be whatever the developer of the plug-in wishes (probably the name of the application and (or) the name of the company that developed the application).
- MenuItemText (required string value) - The text that will appear in the CA Gen workstation under the Plug-in menu.
- ExeLocation (required string value) - The full path and name of the executable to be run when the menu item is selected.
- StartTran (optional string value) - For CA Gen created plug-in applications, this will be the trancode that will be passed to the load module when the plug-in is executed. If this value is not specified, then "XXX" will be passed as the trancode.
- Modal (optional string value) – This sting value must be manually created. If this string value is set to Yes or is not specified, the Toolset instance will be locked when a plug-in application is open. If this string value is set to No, the Toolset instance will be unlocked when a plug-in application is open.
- OptionalParameters (optional string value) - This text will be added a parameters to the plug-in. The parameters will be passed before passing any other parameters. For example, this may be used with CA Gen applications to turn on trace mode and (or) set the database value.

- EnableInDiagrams (optional key added under the application name key) - Each key associated with this key will have as its name the name of the diagram as is documented below in the Diagram Names section. If no keys are specified under this key or this key is not there, "All" is assumed.
- EnableSelectedObjectCount (optional string value) - This value will be associated with each diagram key under the "EnableInDiagrams" key and will be either 0, 1, 2, "*" (for 0 or more selected), "?" (for 0 or 1 selected), or "+" (for 1 or more selected). This will indicate that the menu item will be enabled when the selected object count matches the count given here. If this value is not specified, then the menu item will always be enabled and all selected object id parameters will be passed to the plug-in application.
- EnableSelectedObjectTypes (optional key added under a diagram name key) - Each key associated with this key will have as its name the name of a CA Gen metamodel object type. The names may be found in the Object Decomposition Report which may be accessed through the compiled help file "odrpta.chm". The name will always start with a "T" and will never be more than eight (8) characters in length. If prefixing a "T" will make the name longer than eight characters, then the last character is dropped. For example, to specify a high level entity type object, the metamodel object name is "HLENT" and the text for the key will be "THLENT". For the metamodel object "ACBLKGRP", the text for the key will be "TACBLKGR". Associated with each object type key may be the string value "EnableSelectedObjectCount" as described above. If no value is specified for a specific object type key, then the value specified on the "EnableInDiagrams" key will be used. If more than one object type key is specified, the menu item is enabled when all object types are selected. For example, consider the following registry key hierarchy:

```
\EnableInDiagrams
        \DataModelDiagram
                \THLENT
                        EnableSelectedObjectCount='2'
                         \TRELMM
                           EnableSelectedObjectCount='1'
```

The plug-in menu item would only be enabled in the Data Model Diagram when there are two entities and one relationship selected.

The following is an example of what the registry menu hierarchy might look like for a single plug-in. The example is for a plug-in generated by CA Gen that will change the business system in which an object is located. It allows one or more procedures to be selected when the Dialog Design Diagram is open and works on the current action diagram (as long as it is a BSD action diagram) when that tool is open. It also tells the plug-in that it is not using a database and to run trace.

```
HKEY_CURRENT_USER\Software\ComputerAssociates\CA Gen
\Plug-ins
        \QAT
MenuItemText="Change Business System"
ExeLocation="c:\Program Files\CA\changebs.exe"
OptionalParameters="/db= /test"
\EnableInDiagrams
        \ActionDiagram
                        EnableSelectedObjectCount="0"
                \DialogDesign
                        EnableSelectedObjectCount="+"
```

CA Gen Toolset Menu Hierarchy

Plug-in menu items will be added under the Plug-in menu. The order of the plug-ins on this menu will be the order of the plug-ins folders in the registry.

Plug-in Application Parameters

There will be one or more parameters passed to the plug-in application when it is started from the CA Gen toolset. The full set of parameters will be enclosed in a pair of double-quotes and each parameter will be separated from the others by spaces. The parameters are:

- The name of the diagram from which the plug-in application was called. This parameter will always be the first parameter passed. The names are listed in the Diagram Names section.
- The object id of the main object for that diagram (if one is selected). This parameter will be indicated with a "-m" flag. If there is no main object, the value of this parameter will be -65536.
- Zero or more object ids of any selected objects. Each of these parameters will be indicated with "-o" flags. Note that in the Action Diagram, if lines are selected, these objects will be statement objects only. If there are individual portions of a line selected, the objects will include statement fragments as well as statement objects.

For example, the following might be passed as parameters to a plug-in application:

```
"DialogDesign -m 123456 -o 456789"
```

The example indicates that the Dialog Design diagram is open, the business system is object id 123456, and the selected procedure or procedure step is object id 456789.

Diagram Names

The following is a list of the names that may be passed as the first parameter to a plug-in application. These are also the names that are used in the registry to specify "EnableInDiagrams". Note that if the application is a CA Gen application, the name will come across in upper-case, not mixed-case, but is always specified in mixed case in the registry:

- ActionBlockStructure
- ActionBlockUsage
- ActionDiagram
- ActivityDependancy
- ActivityHierarchy
- AttributeDiagram
- BusinessSystemDesign
- ComponentArchitectureDiagram
- ComponentImplementationDiagram
- ComponentSpecificationDiagram
- DataModelBrowser
- DataModelDiagram
- DataModelList
- DataStoreList
- DataStructureList
- DefineBusinessSystem
- DialogDesignDiagram
- DialogFlowBrowser
- EntityStateTransition
- EnvironmentList
- FlowMaintenance
- Generation
- InterfaceTypeModelDiagram
- Main
- MatrixProcessor
- NavigationDiagramLowerLeft

- NavigationDiagramLowerRight
- NavigationDiagramTop
- OperationDiagram
- OrganizationalHierarchy
- Packaging
- PartitionList
- ProcedureSynthesis
- Prototyping
- Reporting
- ScopedTypeModelDiagram
- ScreenDesigner
- TechnicalDesign
- UnformattedInput
- ViewMaintenance
- WindowInteractionDiagram
- WorkSetList

Using the Toolset Automation Object for Plug-ins

The following steps are used to get to the toolset automation objects that are used to implement plug-ins:

- Fetch the toolset automation object using CreateObject
- Fetch the currently opened model using the Models property of the toolset automation object and the Item method of the Models object (the item will be 1)
- Fetch the OpenAPI object using the OpenAPI property of the Model object. The OpenAPI object contains many of the encyclopedia API functions that may be used by plug-in applications.

The following is a CA Gen action diagram for a plug-in application that performs the above steps and also fetches a count of all the objects in the model and a MetaModelObjects collection object pointing to all the objects in the model. Note that plug-in applications are only executed after a model has already been opened and the views for the objects were all defined with the "Initialize on every entry" checkbox unchecked so that the values may be reused between event calls.

```
--- PLUGIN
| IMPORTS:
| EXPORTS:
| LOCALS:
| Work View toolset work
| obj
| Work View model work
| obj
| Work View openapi work
| obj
| Work View collection work
| obj
| Work View local ief_supplied
| command
| count
| ENTITY ACTIONS:
|
---
--- EVENT ACTION open_window
|
| NOTE ****************************************************
| Fetch the command line parameters (diagram name, scoping
| object, any selected objects)
| Note that you will have to parse the parameters explicitly
| ****************************************************
| SET local ief_supplied command TO COMMAND
|
| NOTE ****************************************************
| Fetch the toolset automation object
| ****************************************************
| SET toolset work obj TO CreateObject("COOLgen.Toolset")
| --- IF toolset work obj IS EQUAL TO NOTHING
| |
|<--|-- ESCAPE
| ---
|
| NOTE ****************************************************
| Fetch the Model object for the currently opened model
| ****************************************************
| SET model work obj TO toolset work obj.Models.Item(1)
| --- IF model work obj IS EQUAL TO NOTHING
| |
|<--|-- ESCAPE
| ---
|
| NOTE ****************************************************
| Fetch the OpenAPI object
```

```
| ********************************************************
| SET openapi work obj TO model work obj.OpenAPI
| --- IF openapi work obj IS EQUAL TO NOTHING
| |
|<--|-- ESCAPE
| ---
|
| NOTE ****************************************************
| Fetch a count of the number of objects in the model
| ********************************************************
| SET local ief_supplied count TO openapi work obj.CountModelObjs
|
| --- IF openapi work obj.LastReturnCode IS NOT EQUAL TO 0
| |
| | NOTE ************************************************
| | An error occurred
| | ************************************************
| ---
|
| NOTE ****************************************************
| Fetch a MetaModelObjects collection for all objects in the model
| ********************************************************
| SET collection work obj TO openapi work obj.FetchAllModelObjs
|
| --- IF openapi work obj.LastReturnCode IS NOT EQUAL TO 0
| |
| | NOTE ************************************************
| | An error occurred
| | ************************************************
| ---
---
--- EVENT ACTION close_window
|
| NOTE ****************************************************
| Clear out all the objects that had been set previously
| ********************************************************
| SET collection work obj TO NOTHING
| SET openapi work obj TO NOTHING
| SET model work obj TO NOTHING
| SET toolset work obj TO NOTHING
---
```

Caveats

The toolset can only display 20 menu items for plug-in applications at one time. If there are more than 20 applications described in the Windows registry, only the first 20 will be displayed.

When creating or updating or deleting items in the current diagram, you will note that in some diagrams, the display is not updated after the plug-in has completed. The diagram display will be updated when the diagram is exited and re-entered. The refresh routines that execute after a commit only update items in the current diagram that might have been changed through actions taken by other diagrams. If the plug-in changes these items, then a refresh will be done.

When using CA Gen to create plug-in applications, remember that the parameters will be passed to the application through the COMMAND system variable. This variable only holds 80 characters so there will be a limit on the number of selected objects that will be passed to the plug-in application. Also note that the parameters come as a single string that will need to be parsed by the plug-in application.

Return Codes

The following is a list of the return code values that may be returned from the OpenAPI and MetaModelObject methods:

EAPI_SUCCESSFUL_RC = 0

EAPI_MAXIMUM_COUNT_REACHED_RC = 1

EAPI_ENCY_NOT_FOUND_RC = 2

EAPI_CONNECT_FAILED_RC = 3

EAPI_DISCONNECT_FAILED_RC = 4

EAPI_USER_NOT_FOUND_RC = 5

EAPI_USER_NOT_AUTH_RC = 6

EAPI_USER_NOT_LOGGED_ON_RC = 7

EAPI_NOT_CONNECTED_RC = 8

EAPI_INCORRECT_PASSWORD_RC = 9

EAPI_MODEL_NOT_FOUND_RC = 10

EAPI_MODEL_ALREADY_LOCKED_RC = 11

EAPI_LOCK_NOT_FOUND_RC = 12

EAPI_OBJECT_NOT_FOUND_RC = 13

EAPI_PROP_NOT_FOUND_RC = 14

EAPI_PTC_INVALID_RC = 15

EAPI_ATC_INVALID_RC = 16

EAPI_OTC_INVALID_RC = 17

EAPI_MNEMONIC_INVALID_RC = 18

EAPI_DEST_OBJ_NOT_FOUND_RC = 19

EAPI_SOURCE_OBJ_NOT_FOUND_RC = 20

EAPI_ASSOC_NOT_FOUND_RC = 21

EAPI_SUBSET_NOT_FOUND_RC = 22

EAPI_OBJECT_NOT_IN_SUBSET_RC = 23

EAPI_OBJECT_NOT_CHECKED_OUT_RC = 24

EAPI_SUBSET_NOT_CHECKED_OUT_RC = 25

EAPI_MODEL_NOT_CHECKED_OUT_RC = 26

EAPI_USER_GROUP_NOT_FOUND_RC = 27

EAPI_GROUP_NOT_FOUND_RC = 28

EAPI_AUTH_NOT_FOUND_RC = 29

EAPI_MODEL_PARENT_NOT_FOUND_RC = 30

EAPI_CHECKOUT_NOT_FOUND_RC = 31

EAPI_MODEL_LOCKED_RC = 32

EAPI_MODEL_INCOMPATIBLE_RC = 33

EAPI_MODEL_FATAL_ERROR_RC = 34

EAPI_DIR_READ_RC = 35

EAPI_BAD_ERRFILE_RC = 36

EAPI_OBJECT_NOT_SHARED_RC = 37

EAPI_MODEL_NOT_LOCKED_RC = 38

EAPI_OBJECT_NOT_ADDED_RC = 39

EAPI_OBJECT_PROTECTED_RC = 40

EAPI_OBJECTS_NOT_IN_SAME_MODEL_RC = 41

EAPI_FROM_OBJECT_NOT_FOUND_RC = 42

EAPI_TO_OBJECT_NOT_FOUND_RC = 43

EAPI_INVALID_CARDINALITY_RC = 44

EAPI_ASSOC_PROTECTED_RC = 45

EAPI_ASSOC1_PROTECTED_RC = 46

EAPI_ASSOC2_PROTECTED_RC = 47

EAPI_OBJECT_1_NOT_FOUND_RC = 48

EAPI_OBJECT_2_NOT_FOUND_RC = 49

EAPI_ASSOC1_NOT_FOUND_RC = 50

EAPI_ASSOC2_NOT_FOUND_RC = 51

EAPI_ASSOC_NOT_ORDERED_RC = 52

EAPI_VALUE_INVALID_RC = 53

EAPI_NOT_AUTHORIZED_RC = 54

EAPI_ASSOC_ALREADY_EXISTS_RC = 55

EAPI_COMMIT_UNSUCCESSFUL_RC = 56

EAPI_OBJECT_NOT_RENAMED = 57

EAPI_DUPLICATE_OBJECT_OBJ_RENAMED = 58

Objects

This section describes the various objects for CA Gen Toolset automation.

OpenAPI Object

An **OpenAPI** object provides functions similar to the Encyclopedia API functions. The **OpenAPI** object is retrieved using the OpenAPI property of the **Model** object.

Properties

LastReturnCode

This is the return code from the last encyclopedia API method executed from the object.

Syntax

object.**LastReturnCode**

Parameters

object

An expression that evaluates to a **OpenAPI** object.

Remarks

The LastReturnCode is a long value whose values are defined in the Plugins chapter of this document.

Methods

AddObject

This method will add an object to the model. It takes an object type code as the single parameter and returns the **MetaModelObject** of the new object that was created.

Syntax

object.**AddObject** *nObjTypeCode*

Parameters

object

An expression that evaluates to an **OpenAPI** object.

nObjTypeCode

A long value indicating the type of object to add.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_OTC_INVALID_RC - if object type code is invalid

Commit

This method will save any changes (since the last commit or rollback) to the transaction file, so that they become a permanent part of the model.

Syntax

object.**Commit**

Parameters

None.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_COMMIT_UNSUCCESSFUL_RC - indicates that the commit did not succeed

CountModelNameTypeObjs

This method will return a count of the number of objects in a model with the given name and type. The parameters are the object type code, the property type code (usually this will be 224 which is the property type code for NAME), and a string for the name to be searched for.

Syntax

object.**CountModelNameTypeObjs** *nObjTypeCode nPrpTypeCode szName*

Parameters

object

An expression that evaluates to an **OpenAPI** object.

nObjTypeCode

A long value indicating the type of object to count.

nPrpTypeCode

A long value indicating the type of the name property. This will usually be the type code for the NAME property.

szName

A string representing the name to be searched for.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_OTC_INVALID_RC - if object type code is invalid

EAPI_PTC_INVALID_RC - if property type code is invalid

CountModelObjs

This method will return a count of the number of objects in a model.

Syntax

object.**CountModelObjs**

Parameters

object

An expression that evaluates to an **OpenAPI** object.

Remarks

EAPI_SUCCESSFUL_RC - if model found

CountModelTypeObjs

This method will return a count of the number of objects of a given type that there are in the model. The only parameter is the object type code.

Syntax

object.**CountModelTypeObjs** *nObjectTypeCode*

Parameters

object

An expression that evaluates to an **OpenAPI** object.

nObjTypeCode

A long value indicating the type of object to count.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_OTC_INVALID_RC - if object type code is invalid

FetchAllModelObjects

This method will fetch all the objects in the current model. It takes no parameters and returns a **MetaModelObjects** collection object.

Syntax

object.**FetchAllModelObjs**

Parameters

object

An expression that evaluates to an **OpenAPI** object.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

FetchAllModelTypeObjs

This method will fetch all the objects of a given type in the current model. The only parameter is the object type code and it returns a **MetaModelObjects** collection object.

Syntax

object. **FetchAllModelTypeObjs** nObjTypeCode

Parameters

object

An expression that evaluates to an **OpenAPI** object.

nObjTypeCode

A long value indicating the type of object to fetch.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_OTC_INVALID_RC - if object type code is invalid

FetchModelInfo

Retrieves the model name.

Syntax

object. **FetchModelInfo**

Parameters

object

An expression that evaluates to an **OpenAPI** object.

Remarks

This method always returns **EAPI_SUCCESSFUL_RC**.

FetchModelNameTypeObjs

This method will fetch all the objects in a model with the given name and type. The parameters are the object type code, the property type code (usually this will be 224 which is the property type code for NAME), and a string for the name to be searched for. It returns a **MetaModelObjects** collection object.

Syntax

object.**FetchModelNameTypeObjs** *nObjTypeCode nPrpTypeCode szName*

Parameters

bject

An expression that evaluates to an **OpenAPI** object.

nObjTypeCode

A long value indicating the type of object to count.

nPrpTypeCode

A long value indicating the type of the name property. This will usually be the type code for the NAME property.

szName

A string representing the name to be searched for.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_OTC_INVALID_RC - if object type code is invalid

EAPI_PTC_INVALID_RC - if property type code is invalid

FetchSchemaAscTypeByMnem

This method returns the association type code associated with the given mnemonic. The only parameter is the mnemonic for the association.

Syntax

object.**FetchSchemaAscTypeByMnem** *pszMnemonic*

Parameters

pszMnemonic

A string value representing the association mnemonic.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_MNEMONIC_INVALID_RC - if mnemonic is invalid

FetchSchemaObjTypeByMnem

This method returns the object type code associated with the given mnemonic. The only parameter is the mnemonic for the object. Note that object mnemonics always start with the letter "T" and may be no more than eight (8) characters in length. If the mnemonic would be longer than eight characters with the addition of a "T" on the front, then the last character of the mnemonic is dropped.

Syntax

object.**FetchSchemaObjTypeByMnem** *pszMnemonic*

Parameters

pszMnemonic

A string value representing the object mnemonic.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_MNEMONIC_INVALID_RC - if mnemonic is invalid

FetchSchemaPrpTypeByMnem

This method returns the property type code associated with the given mnemonic. The only parameter is the mnemonic for the parameter.

Syntax

object.**FetchSchemaPrpTypeByMnem** *pszMnemonic*

Parameters

pszMnemonic

A string value representing the property mnemonic.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_MNEMONIC_INVALID_RC - if mnemonic is invalid

GetMetaModelObject

This method will fetch an object in the current model. The only parameter is the object id number. It returns a **MetaModelObject**.

Syntax

bject.**GetMetaModelObject** nObjId

Parameters

object

An expression that evaluates to an OpenApi object.

nObjId

A long value indicating the id of the object to fetch.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_OBJECT_NOT_FOUND_RC - no object exists with the given object id

Rollback

This method will cause any changes since the last commit to not be saved in the transaction file.

Note: This does not completely undo all changes to the working model. Do not save the model on the local system after a rollback.

Syntax

object.**UpRollback**

Parameters

None.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

MetaModelObject

A **MetaModelObject** is an object that points to a single meta-model object. It contains many methods that are similar to the Encyclopedia API functions.

Properties

ID

This is the object id of the meta-model object. Note that the object id is not the same as the CEID property value.

Syntax

object.**ID**

Parameters

object

An expression that evaluates to a **MetaModelObject**.

ObjectTypeCode

This is the object type code of the meta-model object.

Syntax

object.**ObjectTypeCode**

Parameters

object

An expression that evaluates to a **MetaModelObject**.

Remarks

The **ObjectTypeCode** is a long numeric value that indicates the type of the meta-model object.

LastReturnCode

This is the return code from the last encyclopedia API method executed from the object

Syntax

object.**LastReturnCode**

Parameters

object

An expression that evaluates to a **MetaModelObject**.

Remarks

The LastReturnCode is a long value whose values are defined in the Plugins chapter of this document.

Methods

AddAssoc

This method creates an association between the current object and a given object. The parameters are the association type code and the destination object id.

Syntax

object.**AddAssoc** *nAscTypeCode nDestObjID*

Parameters

object

An expression that evaluates to a **MetaModelObject**.

nAscTypeCode

A long value indicating the type of association to add.

nDestObjID

A long value indicating the object id of the destination object.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_ATC_INVALID_RC - if association type code is invalid

EAPI_FROM_OBJECT_NOT_FOUND_RC - indicates that the current object is invalid

EAPI_TO_OBJECT_NOT_FOUND_RC - indicates that the destination object is invalid

EAPI_ASSOC_NOT_FOUND_RC - indicates that the association is not valid between the two objects

EAPI_ASSOC_ALREADY_EXISTS_RC - indicates that the association being added already exists

EAPI_ASSOC_PROTECTED_RC - indicates that adding this association would cause a protection error

CountCardManyAsc

This method returns a count of the number of objects that are associated with a the current object using a given association. The only parameter is the association type code.

Syntax

object.**CountCardManyAsc** *nAscTypeCode*

Parameters

object

An expression that evaluates to a **MetaModelObject**.

nAscTypeCode

A long value indicating the type of association to follow.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_ATC_INVALID_RC - if object type code is invalid

EAPI_OBJECT_NOT_FOUND_RC - indicates that the current object is invalid

DelAssoc

This method will delete an association between the current object and a given object. The parameters are the association type code, the ID of the target object, and a flag indicating whether a trigger delete should be executed or not. It returns the LastReturnCode value.

Syntax

LastReturnCode = object.**DelAssoc** *nAscTypeCode nToObjID nTriggerFlag*

Parameters

object

An expression that evaluates to a **MetaModelObject**.

LastReturnCode

A long value representing the return code from the method.

nAscTypeCode

A long value indicating the type of association to follow.

nToObjID

A long value indicating the object id of the target object.

nTriggerFlag

A boolean value used to tell the delete method to trigger delete related objects when this association is deleted. A zero value indicates that a non-trigger delete will be executed (only the association will be deleted) and a non-zero value indicates that a trigger delete will be executed (the association and any objects related through mandatory associations will be deleted).

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_FROM_OBJECT_NOT_FOUND_RC - indicates that the current object is invalid

EAPI_TO_OBJECT_NOT_FOUND_RC - indicates that the target object is invalid

EAPI_ATC_INVALID_RC - indicates that the object type code passed in is invalid

EAPI_ASSOC_NOT_FOUND_RC - indicates that the association is not valid between the two objects

EAPI_ASSOC_PROTECTED_RC - indicates that the deletion of the association would cause a protection error

DelObject

This method will delete the current object from the model. It takes one parameter (a flag indicating whether a trigger delete should be executed or not) and returns the LastReturnCode value.

Syntax

LastReturnCode = object.**DelObject** *nTriggerFlag*

Parameters

object

An expression that evaluates to a **MetaModelObject**.

LastReturnCode

A long value representing the return code from the method.

nTriggerFlag

A boolean value used to tell the delete method to trigger delete related objects when this object is deleted. A zero value indicates that a non-trigger delete will be executed (only the object will be deleted) and a non-zero value indicates that a trigger delete will be executed (the object and any objects related to it through mandatory associations will be deleted).

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_OBJECT_NOT_FOUND_RC - indicates that the current object is invalid

EAPI_OBJECT_PROTECTED_RC - indicates that the deletion of the current object would cause a protection error

FetchCardManyAsc

This method will fetch all objects associated with the current object through a given association. The only parameter is the association type code and it returns a **MetaModelObjects** collection object.

Syntax

object.**FetchCardManyAsc** *nAscTypeCode*

Parameters

object

An expression that evaluates to a **MetaModelObject**.

nAscTypeCode

A long value indicating the type of association to follow.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_ATC_INVALID_RC - indicates that the object type code passed in is invalid

EAPI_ASSOC_NOT_FOUND_RC - indicates that the association is not valid for the current object

EAPI_OBJECT_NOT_FOUND_RC - indicates that the current object is invalid

EAPI_DEST_OBJ_NOT_FOUND_RC - indicates that the association does not exist

FetchCardOneAsc

This method will fetch the object associated with the current object through a given association. The only parameter is the association type code and it returns a **MetaModelObject** object for the target object.

Syntax

object.**FetchCardOneAsc** *nAscTypeCode*

Parameters

object

An expression that evaluates to a **MetaModelObject**.

nAscTypeCode

A long value indicating the type of association to follow.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_ATC_INVALID_RC - indicates that the object type code is either not a cardinality one association or not a valid association type code

EAPI_ASSOC_NOT_FOUND_RC - indicates that the association is not valid for the current object

EAPI_FROM_OBJECT_NOT_FOUND_RC - indicates that the current object is invalid

EAPI_DEST_OBJ_NOT_FOUND_RC - indicates that the association does not exist

FetchCharArrayPrp

This method will fetch the value of a string property for the current object. It takes one parameter, the property type code for the string property.

Syntax

object.**FetchCharArrayPrp** *nPrpTypeCode*

Parameters

object

An expression that evaluates to a **MetaModelObject**.

nPrpTypeCode

A long value indicating the type of the string property.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_PTC_INVALID_RC - indicates that the property type code is either not a valid code, does not exist for the current object, or is not a NAME, LOADNAME, DESC, or STRING property

EAPI_OBJECT_NOT_FOUND_RC - indicates that the current object is invalid

FetchSingleCharPrp

This method will fetch the value of a single character property for the current object. It takes one parameter, the property type code for the character property.

Syntax

object.**FetchSingleCharPrp** *nPrpTypeCode*

Parameters

object

An expression that evaluates to a **MetaModelObject**.

nPrpTypeCode

A long value indicating the type of the character property.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_PTC_INVALID_RC - indicates that the property type code is either not a valid code, does not exist for the current object, or is not a CHAR property

EAPI_OBJECT_NOT_FOUND_RC - indicates that the current object is invalid

FetchSingleNumericPrp

This method will fetch the value of a numeric property for the current object. It takes one parameter, the property type code for the numeric property.

Syntax

object.**FetchSingleNumericPrp** *nPrpTypeCode*

Parameters

object

An expression that evaluates to a **MetaModelObject**.

nPrpTypeCode

A long value indicating the type of the numeric property.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_PTC_INVALID_RC - indicates that the property type code is either not a valid code, does not exist for the current object, or is not a numeric property

EAPI_OBJECT_NOT_FOUND_RC - indicates that the current object is invalid

ReordrAssocAftr

Given two objects that are related to the current object through a given ordered association, change the order of the two objects so that the second object comes after the first. The method returns the LastReturnCode.

Syntax

LastReturnCode = object.**ReordrAssocAftr** *nAscTypeCode nToObjID1 nToObjID2*

Parameters

object

An expression that evaluates to a **MetaModelObject**.

LastReturnCode

A long value representing the return code from the method.

nAscTypeCode

A long value indicating the type of association to follow.

nToObjID1

A long value indicating the object id of the first target object.

nToObjID2

A long value indicating the object id of the second target object.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_FROM_OBJECT_NOT_FOUND_RC - indicates that the current object is invalid

EAPI_OBJECT_1_NOT_FOUND_RC - indicates that the first object is invalid

EAPI_OBJECT_2_NOT_FOUND_RC - indicates that the second object is invalid

EAPI_ATC_INVALID_RC - indicates that the association type code is invalid

EAPI_ASSOC1_PROTECTED_RC - indicates that the association from the current object to the first object is protected

EAPI_ASSOC2_PROTECTED_RC - indicates that the association from the current object to the second object is protected

EAPI_ASSOC1_NOT_FOUND_RC - indicates that the association from the current object to the first object does not exist

EAPI_ASSOC2_NOT_FOUND_RC- indicates that the association from the current object to the second object does not exist

EAPI_ASSOC_NOT_ORDERED_RC - indicates that the association is not ordered

ReordrAssocBefr

Given two objects that are related to the current object through a given ordered association, change the order of the two objects so that the second object comes before the first. The method returns the LastReturnCode.

Syntax

LastReturnCode = object.**ReordrAssocBefr** *nAscTypeCode nToObjID1 nToObjID2*

Parameters

object

An expression that evaluates to a **MetaModelObject**.

LastReturnCode

A long value representing the return code from the method.

nAscTypeCode

A long value indicating the type of association to follow.

nToObjID1

A long value indicating the object id of the first target object.

nToObjID2

A long value indicating the object id of the second target object.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_FROM_OBJECT_NOT_FOUND_RC - indicates that the current object is invalid

EAPI_OBJECT_1_NOT_FOUND_RC - indicates that the first object is invalid

EAPI_OBJECT_2_NOT_FOUND_RC - indicates that the second object is invalid

EAPI_ATC_INVALID_RC - indicates that the association type code is invalid

EAPI_ASSOC1_PROTECTED_RC - indicates that the association from the current object to the first object is protected

EAPI_ASSOC2_PROTECTED_RC - indicates that the association from the current object to the second object is protected

EAPI_ASSOC1_NOT_FOUND_RC - indicates that the association from the current object to the first object does not exist

EAPI_ASSOC2_NOT_FOUND_RC- indicates that the association from the current object to the second object does not exist

EAPI_ASSOC_NOT_ORDERED_RC - indicates that the association is not ordered

UpCharArray

This method will update a string property value for the current object. The method returns the LastReturnCode.

Syntax

LastReturnCode = object.**UpCharArray** *nPrpTypeCode*

Parameters

object

An expression that evaluates to a **MetaModelObject**.

LastReturnCode

A long value representing the return code from the method.

nPrpTypeCode

A long value indicating the type of the string property.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_PTC_INVALID_RC - indicates that the property type code is either not a valid code, does not exist for the current object, or is not a NAME, LOADNAME, DESC, or STRING property

EAPI_OBJECT_NOT_FOUND_RC - indicates that the current object is invalid

UpNumeric

This method will update a numeric property value for the current object. The method returns the LastReturnCode.

Syntax

LastReturnCode = object.**UpNumeric** *nPrpTypeCode*

Parameters

object

An expression that evaluates to a **MetaModelObject**.

LastReturnCode

A long value representing the return code from the method.

nPrpTypeCode

A long value indicating the type of the numeric property.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_PTC_INVALID_RC - indicates that the property type code is either not a valid code, does not exist for the current object, or is not a numeric property

EAPI_OBJECT_NOT_FOUND_RC - indicates that the current object is invalid

UpSingleCharArray

This method will update a single character property value for the current object. The method returns the LastReturnCode.

Syntax

LastReturnCode = object.**UpSingleCharArray** *nPrpTypeCode*

Parameters

object

An expression that evaluates to a **MetaModelObject**.

LastReturnCode

A long value representing the return code from the method.

nPrpTypeCode

A long value indicating the type of the character property.

Remarks

EAPI_SUCCESSFUL_RC - indicates a successful execution

EAPI_PTC_INVALID_RC - indicates that the property type code is either not a valid code, does not exist for the current object, or is not a CHAR property

EAPI_OBJECT_NOT_FOUND_RC - indicates that the current object is invalid

MetaModelObjects

A **MetaModelObjects** object is an object that points to a single collection of MetaModelObject objects. It contains one property and one method that are used to access the individual **MetaModelObject** objects.

Properties

count

This is a count of the number of **MetaModelObject** objects that are available to this **MetaModelObjects** collection object.

Syntax

object.**count**

Parameters

object

An expression that evaluates to a **MetaModelObjects** collection object.

Methods

Item

This method returns the MetaModelObject associated with the given index. Note that the index is one-based.

Syntax

object.**Item** *index*

Parameters

object

An expression that evaluates to a **MetaModelObjects** collection object.

index

A long value indicating which **MetaModelObject** to return.

Remarks

If the index is out of range, the LastReturnCode of the **MetaModelObject** will be set to **EAPI_OBJECT_NOT_FOUND_RC**. Otherwise, it will be set to **EAPI_SUCCESSFUL_RC**.

Model Object

The **Model** object represents an open CA Gen model.

Properties

Active

Gets or sets whether model is active.

Syntax

object.**Active** [=*boolean*]

Parameters

object

An expression that evaluates to a model object.

boolean

A **Boolean** that sets the state of the model. Possible values are:

- **True** - Activates the model.
- **False** - Deactivates the model.

Return Values

The **Active** property returns one of the following values:

- **True** - The model is active.
- **False** - The model is not active.

Remarks

The **Active** property has the **Boolean** type.

Since the Toolset only allows one model to be opened at any time, this function is useful. However, it does allow for future changes. The current implementation will never deactivate a model and the return value will always be True.

ActiveWindow

Gets the Window object associated with the active window.

Syntax

object.**ActiveWindow**

Parameters

object

An expression that evaluates to a model object.

FullName

Gets the full model path.

Syntax

object.**FullName**

Parameters

object

An expression that evaluates to a model object.

Remarks

The **FullName** property has the **String** type.

The string returned will contain the full path to the directory containing the model files.

Name

Gets the local model name.

Syntax

object.**Name**

Parameters

object

An expression that evaluates to a model object.

Remarks

The **Name** property has the **String** type.

The string returned will contain the local model name for the model.

OpenAPI

Gets an **OpenAPI** object.

Syntax

object.**OpenAPI**

Parameters

object

An expression that evaluates to a model object.

Remarks

The **OpenAPI** property is an **OpenAPI** type.

Parent

Gets the parent of the model object.

Syntax

object.**Parent**

Parameters

object

An expression that evaluates to a model object.

Remarks

The model object will return the Toolset object for a parent.

Path

Gets the path to a model. This path never ends with a backslash, unless the path has the format "C:\."

Syntax

object.**Path**

Parameters

object

An expression that evaluates to a model object.

Remarks

The **Path** property has the **String** type.

The **Path** property does not get a model's file name and extension. To get the file name and extension, use the Name or FullName property.

ReadOnly

Gets the read-only state of the model.

Syntax

object. **ReadOnly**

Parameters

object

An expression that evaluates to a model object.

Return Values

The **ReadOnly** property returns one of the following values:

- **True** - The model is read-only.
- **False** - The model is not read-only.

Remarks

The **ReadOnly** property has the **Boolean** type.

Saved

Gets the saved status of the model.

Syntax

object. **Saved** [=*boolean*]

Parameters

object

An expression that evaluates to a model object.

Return Values

The **Saved** property returns one of the following values:

- **True** - The model changes are saved.
- **False** - The model changes are not saved.

Remarks

The **Saved** property has the **Boolean** type.

Toolset

Gets the CA Gen Toolset object.

Syntax

object.**Toolset**

Parameters

object

An expression that evaluates to a **Models** object.

Remarks

The **Toolset** property has the **Toolset** type.

The **Toolset** object represents the entire CA Gen application. It provides access to other objects in the CA Gen object model and provides methods and properties that affect the entire application.

Methods

Tools

Gets the Tools object.

Syntax

object.**Tools**

Parameters

object

An expression that evaluates to a model object.

Remarks

The **Tools** property has the **Tools** type.

The **Tools** object is a collection object that represents all available tools associated with the model.

Each tool in this collection is represented by a Tool object.

Windows

Gets the Windows object.

Syntax

object.**Windows**

Parameters

object

An expression that evaluates to a model object.

Remarks

The **Windows** property has the **Windows** type.

The **Windows** object is a collection object that represents all open windows associated with the model.

Each window in this collection is represented by a Window object.

Models Object

The **Models** object represents the collection of open models. Each model in the collection is represented by a Model object. Currently, the CA Gen toolset only supports one open model. The Models collection will always contain zero or one Model object. The Models object supports our current implementation while allowing for future changes without changes to the interface.

Properties

_NewEnum

References objects in a collection.

Syntax

object.**_NewEnum**

Parameters

object

An expression that evaluates to a Models object.

Remarks

With Visual C++, you can browse a collection to find a particular item by using the **_NewEnum** property or the Item method. In Visual Basic, you do not need to use the **_NewEnum** property, because it is automatically used in the implementation of **For Each ... Next**.

Toolset

Gets the CA Gen Toolset object.

Syntax

object.**Toolset**

Parameters

object

An expression that evaluates to a **Models** object.

Remarks

The **Toolset** property has the **Toolset** type.

The **Toolset** object represents the entire CA Gen application. It provides access to other objects in the CA Gen object model and provides methods and properties that affect the entire application.

Count

Gets the number of items in the models collection.

Syntax

object.**Count**

Parameters

object

An expression that evaluates to a **Models** object.

Remarks

The **Count** property has the **Long** type.

Parent

Gets the parent of a models object.

Syntax

object.**Parent**

Parameters

object

An expression that evaluates to a **Models** object.

Remarks

The Toolset object is the parent object for the Models object.

Methods

Add

Creates or opens a model.

Syntax

object.**Add** *modelName pathName*

Parameters

object

An expression that evaluates to a **Models** object.

ModelName

A String that represents the long model name for the model

PathName

A **String** that represents the location for the model. If the model name is given without a directory location, the model is created in the modelPath directory.

Remarks

If a model exists at Pathname, the model is opened. Otherwise, a model is created and added to the Models collection.

CloseAll

Closes all models.

Syntax

object.**CloseAll** [*SaveChanges*]

Parameters

object

An expression that evaluates to a **Models** object.

SaveChanges

(Optional) An enum of type **DsSaveChanges** that indicates whether to save changes to one or more documents before closing them. Following are the possible values, which have the **Long** type:

dsSaveChangesYes - Saves the changes (the default).

dsSaveChangesNo - Does not save the changes.

dsSaveChangesPrompt - Prompts the user to save changes. If the user chooses not to save the changes, **CloseAll** returns the enum **dsSaveStatus** with the value **dsSaveCanceled**.

Not Yet Implemented

Copy

Gets the parent of a models object.

Syntax

object.**SourcePath DestinationPath ModelName**

Parameters

object

An expression that evaluates to a **Models** object.

SourcePath

A **String** that represents the source path for a model.

DestinationPath

A **String** that represents the destination path for the copied model.

ModelName

A **String** that represents the new long ModelName.

Remarks

ModelName must be 32-characters or less.

Not Yet Implemented

Delete

Deletes a model.

Syntax

object.**SourcePath**

Parameters

object

An expression that evaluates to a **Models** object.

SourcePath

A **String** that represents the source path for a model.

Not Yet Implemented

Item

Gets a specified model from a collection.

Syntax

object.**Item** [*index*]

-or-

object index

Parameters

object

An expression that evaluates to a **Models** object.

index

A **Variant** that is a **Long** or **String** representing model.

- If you specify a **Long**, the **Item** method fetches the object by its one-based index in the collection.
- If you specify a **String**, it must be the model's long model name.

New

Creates a model with a specified file name.

Syntax

object.**New** *PathName LongName*

Parameters

object

An expression that evaluates to a **Models** object.

PathName

A **String** that specifies the full path to the model.

LongName

A String that specifies the long model name.

Open

Opens a model with a specified file name.

Syntax

object.**Open** *PathName*

Parameters

object

An expression that evaluates to a **Models** object.

PathName

A **String** that specifies the full path to the model.

SaveAll

Saves all open models in the collection.

Syntax

object.**SaveAll**

Parameters

object

An expression that evaluates to a **Models** object.

CopyWithSubstitution Object

A **CopyWithSubstitution** is a specific tool type that provides copy with substitution capabilities.

Properties

Toolset

Gets the CA Gen Toolset object.

Syntax

object.**Toolset**

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Remarks

The **Toolset** property has the **Toolset** type.

The **Toolset** object represents the entire CA Gen application. It provides access to other objects in the CA Gen object model and provides methods and properties that affect the entire Toolset.

Parent

Gets the parent object.

Syntax

object.**Parent**

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Remarks

The **Parent** property's type for the Tool object is the Model object.

Type

Gets the specific tool type.

Syntax

object.**Type**

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Remarks

The **Tool** property has the type **dsToolType**.

TS_TOOLTYPE is a Enum type. Possible values are:

TS_TOOLTYPE_COPYWITHSUBSTITUTION - Copy With Substitution

DestinationBusinessSystem

Gets or Sets name of destination Business System.

Syntax

object.**DestinationBusinessSystem** = [Business system]

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Remarks

Business system

A Variant that is a Long or String representing a valid model object Business System:

- If you specifiy a **Long**, the **DestinationBusinessSystem** method sets the object id as a business system object id.
- If you specify a **String**, it must be a valid Business System name.

A **long** is returned as the **DestinationBusinessSystem** Object ID.

DestinationProcedure

Gets or Sets name of destination procedure step.

Syntax

object. **DestinationProcedure** = [Procedure Step]

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Remarks

Procedure
A String that represents a new procedure step.

A String is returned as the **DestinationProcedure** name.

SourceProcedure

Gets or Sets name of source procedure step.

Syntax

object. **SourceProcedure** = [Procedure Step]

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Remarks

Procedure

A long that represents a valid model object procedure step. This will be the original procedure that is copied to a destination procedure.

A long is returned as the **SourceProcedure** Object ID.

IncludeAllWindows

Gets or Sets option to Include all Source Windows in copy.

Syntax

object. **SourceProcedure** = [Boolean]

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Remarks

Boolean

True if source windows should be included in copy. False if source windows should not be copied.

A boolean is returned as the current option setting. False is the default setting.

Methods

GetDestinationBusinessSystems

Gets available destination business system objects for the destination procedure.

Syntax

object. **GetDestinationBusinessSystems**

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Remarks

pBSysIDs - [out] long *

Array of business systems.

GetSourceBusinessSystems

Gets available source business system objects.

Syntax

object. **GetSourceBusinessSystems**

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Remarks

pBSysIDs - [out] long *

Array of business systems.

GetMatchedObject

Gets object id that has been matched with the specified object id.

Syntax

ID2 = *object*. **GetMatchedObject ID1**

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

ID1 - [in] long

An object id that evaluates to a specific object id.

ID2 - [out] long

An object that has been matched with object ID1.

Remarks

A value of 0 will be returned if object ID1 is not in source list or has not been matched.

GetObjectName

Gets name for an object id.

Syntax

object. **GetObjectName** ID name

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

ID - [in] long

An object id that evaluates to a specific object id.

Remarks

A **String** is returned represent the name for the object id.

GetSourceProcedures

Gets procedures steps available for the **SourceProcedure** property.

Syntax

object. **GetSourceProcedures** BusinessSystem

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

BusinessSystem - [in] long

An object id that evaluates to a specific Business System type. If business system is 0, the root business system is used.

Remarks

An array of **long** values are returned representing the procedure object ids for the given business system.

GetDestinationEntityTypes

Gets available entity types for a given subject area.

Syntax

object. **GetDestinationEntityTypes** SubjectArea

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

SubjectArea - [in] long

An object id that evaluates to a specific Subject Area type.

Remarks

An array of **long** values are returned representing the entity types for the given subject area.

GetSourceEntityTypes

Gets available entity types for a given subject area.

Syntax

object. **GetSourceEntityTypes**

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Remarks

An array of **long** values are returned representing the entity types for the source procedure.

Match

Matches entities, attributes, and relationships.

Syntax

object. Match SourceID DestinationID MatchAttributes

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

SourceID - [in] long

An object id that evaluates to a specific source object id (entity type, attribute, relationship).

DestinationID -[in] long

An object id that evaluates to a specific destination object id (entity type, attribute, relationship).

MatchAttributes - [in] bool

Boolean to indicate if associated attributes should be matched. Only applies if SourceID and DestinationID are entity types.

Copy

Performs Copy with Substitution based on source and destination objects.

Syntax

object. **Copy**

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

GetDestinationRelationships

Gets available relationships for a destination entity type.

Syntax

object. **GetRelationships** Entity

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Entity - [in] long

An object id that evaluates to a specific Entity type.

Remarks

An array of **long** values are returned representing the relationship types for the given entity.

GetSourceRelationships

Gets available relationships for a source entity type.

Syntax

object. **GetRelationships** Entity

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Entity - [in] long

An object id that evaluates to a specific Entity type.

Remarks

An array of **long** values are returned representing the relationship types for the given entity.

GetDestinationAttributes

Gets attributes for a given entity object.

Syntax

object. **GetAttributes** Entity

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Entity - [in] long

An object id that evaluates to a specific Entity object id.

Remarks

An array of **long** values are returned representing the attribute types for the given entity.

GetSourceAttributes

Gets attributes for a given entity object.

Syntax

object. **GetSourceAttributes** Entity

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Entity - [in] long

An object id that evaluates to a specific Entity object id.

Remarks

An array of **long** values are returned representing the attribute types for the given entity.

Unmatch

Removes matches of entities, attributes, and relationships.

Syntax

object. **Unmatch** SourceID DestinationID

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

SourceID - [in] long

An object id that evaluates to a specific object id (entity type, attribute, relationship).

DestinationID -[in] long

An object id that evaluates to a specific object id (entity type, attribute, relationship).

GetDestinationSubjectAreas

Gets available subject areas.

Syntax

object. **GetSourceSubjectAreas**

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Long

A subject area that represents the parent subject area. A value of 0 represents the root subject area.

Remarks

An array of **long** values are returned representing the subject areas for the given subject area.

GetObjectTypeCode

Gets numeric type of given model.

Syntax

object. **GetObjectTypeCode** ID

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

ID- [in] long

An object id that evaluates to a specific object id.

Remarks

A **long** value is returned representing the type of object.

GetSourceActionBlocks

Gets available action blocks.

Syntax

object. **GetSourceActionBlocks** actionBlocks

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

Remarks

An array of action blocks are returned.

GetRelationshipTo

Gets destination object that is pointed to by the relationship object.

Syntax

object. **GetRelationshipTo** relationshipID

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

RelationshipID

A long representing a relationship object.

Remarks

An object ID representing the object the relationship goes to.

GetRelationshipFrom

Gets source object that is pointed to by the relationship object.

Syntax

object. **GetRelationshipFrom** relationshipID

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

RelationshipID

A long representing a relationship object.

Remarks

An object ID representing the object the relationship originates from.

GetInverseRelationship

Gets relationship object that represents to relationship in the other direction from the given relationship.

Syntax

object. **GetInverseRelationship** relationshipID

Parameters

object

An expression that evaluates to a **CopyWithSubstitution** object.

RelationshipID

A long representing a relationship object.

Remarks

An object ID representing the other inverse relationship from the given relationship.

Tool Object

A **Tool** object is a generic view of a specific tool object. The generic object contains methods and properties common to all tool objects. The specific object contains methods and properties that represent a specific set of functionality in the Toolset Application.

Properties

Toolset

Gets the CA Gen Toolset object.

Syntax

object.**Toolset**

Parameters

object

An expression that evaluates to a **Tool** object.

Remarks

The **Toolset** property has the **Toolset** type.

The **Toolset** object represents the entire CA Gen Toolset. It provides access to other objects in the CA Gen object model and provides methods and properties that affect the entire Toolset.

Parent

Gets the parent object.

Syntax

object.**Parent**

Parameters

object

An expression that evaluates to a **Tool** object.

Remarks

The **Parent** property's type for the Tool object is the Toolset object.

Type

Gets the specific tool type.

Syntax

object.**Type**

Parameters

object

An expression that evaluates to a **Tool** object.

Remarks

The **Tool** property has the type **dsToolType**.

DsToolType is a **Long** type. Possible values are:

DsToolCopyWithSubstitution - Copy With Substitution

Tools Object

The **Tools** object represents a collection of available Tool objects. A Tool represents a specific set of functionality in the Toolset Application.

Properties

_NewEnum

References objects in the **Tools** collection.

Syntax

object.**_NewEnum**

Parameters

object

An expression that evaluates to a **Tools** object.

Remarks

With Visual C++, you can browse a collection to find a particular item by using the **_NewEnum** property or the Item method. In Visual Basic, you do not need to use the **_NewEnum** property, because it is automatically used in the implementation of **For Each ... Next**.

Toolset

Gets the CA Gen **Toolset** object.

Syntax

object.**Toolset**

Parameters

object

An expression that evaluates to a **Tools** object.

Remarks

The **Toolset** property has the **Toolset** type.

The **Toolset** object represents the entire CA Gen Toolset. It provides access to other objects in the CA Gen object model and provides methods and properties that affect the entire Toolset.

Count

Gets the number of tools in the collection.

Syntax

object.**Count**

Parameters

object

An expression that evaluates to a **Tools** object.

Remarks

The **Count** property has the **Long** type.

Parent

Gets the parent object.

Syntax

object.**Parent**

Parameters

object

An expression that evaluates to a Tools object.

Remarks

The **Parent** property's type for the Tools object is the Toolset object.

Methods

Item

Gets a specified tool from the collection.

Syntax

object.**Item** [*index*]

-or-

object index

Parameters

object

An expression that evaluates to a Tools object.

Index

A Variant that is a Long or String representing a tool.

- If you specifiy a **Long**, the Item method fetches the object by its one-based index in the collection.
- If you specify a **String**, it must be the Tool's name.

Toolset Object

The **Toolset** object represents CA Gen and is the topmost object in the object hierarchy. From the Toolset object, you can directly access other objects by using the Toolset object's properties and methods, or you can indirectly access objects through other objects obtained by these properties and methods.

The Toolset object has the following properties, methods, and events:

Properties

ActiveWindow

Gets the Window object associated with the active window.

Syntax

object.**ActiveWindow**

Parameters

object

An expression that evaluates to the Toolset object.

Remarks

The **ActiveWindow** property has the **Window**.

AutoSave

Gets or sets whether the Toolset automatically saves a model at specific intervals.

Syntax

object.**AutoSave** [=*interval*]

Parameters

object

An expression that evaluates to a **Toolset** object.

interval

A **long** that sets the autosave time interval in minutes. A value of zero will disable autosave.

Remarks

Prompt will not appear for autosave.

BackgroundColor

Gets or sets the background color of Toolset windows.

Syntax

object.**BackgroundColor** [=*value*]

Parameters

object

An expression that evaluates to the Toolset object.

value

An OLE_COLOR that sets the background color.

Remarks

The **BackgroundColor** property has the **OLE_COLOR** type.

FullName

Gets the full path.

Syntax

object.**FullName**

Parameters

object

An expression that evaluates to the Toolset object.

Remarks

The **FullName** property has the **String** type.

Height

Gets or sets the height of a window in pixels.

Syntax

object.**Height** [=*value*]

Parameters

object

An expression that evaluates to the Toolset object.

value

A **Long** that sets the height of the window in pixels.

Remarks

The **Height** property has the **Long** type.

The **Height** property gets or sets the distance in pixels between the top and bottom edge of the main Toolset window.

Left

Gets or sets the x-coordinate of a window's left edge in pixels.

Syntax

object.**Left** [=*value*]

Parameters

object

An expression that evaluates to the Toolset object.

value

A **Long** that sets the x-coordinate of the window's left edge in pixels.

Remarks

The **Left** property has the **Long** type.

The **Left** property gets or sets the distance in pixels between the left edge of the screen and the main Toolset window.

ModelPath

Gets or sets the model path.

Syntax

object.**ModelPath**

Parameters

object

An expression that evaluates to the Toolset object.

Remarks

The **ModelPath** property has the **String** type.

Models

Gets the Models object.

Syntax

object.**Models**

Parameters

object

An expression that evaluates to the Toolset object.

Remarks

The **Models** property has the **Models** type.

The **Models** object represents the collection of open models. Each model in the collection is represented by a Model object.

Name

Gets the name of an object.

Syntax

object.**Name**

-or-

object

Parameters
object

object

An expression that evaluates to the Toolset object.

Remarks

The **Name** property has the **String** type. The name of the Toolset is returned.

OutputPath

Gets or sets the output path.

Syntax

object. **OutputPath**

Parameters

object

An expression that evaluates to the Toolset object.

Remarks

The **OutputPath** property has the **String** type.

Parent

Gets the parent of an object.

Syntax

object.**Parent**

Parameters

object

An expression that evaluates to the Toolset object.

Remarks

The **Parent** property's returns the Toolset parent object.

Path

Gets the path to an object. This path never ends with a backslash, unless the path has the format "C:\."

Syntax

*object.***Path**

Parameters

object

An expression that evaluates to the Toolset object.

Remarks

The **Path** property has the **String** type.

The **Path** property does not get an object's file name and extension. To get the file name and extension, use the Name or FullName property. The **Path** property gets the path to the CA Gen toolset executable.

Toolbar

Gets or sets whether the toolbar is displayed.

Syntax

object. **Toolbar** [=*boolean*]

Parameters

object

An expression that evaluates to a **Toolset** object.

boolean

A **Boolean** that sets whether the toolbar is displayed. Possible values are:

- **True** - toolbar is visible
- **False** - toolbar is invisible

Remarks

The **Toolbar** property has the **Boolean** type. *Not Yet Implemented.*

Top

Gets or sets the y-coordinate of a window's top edge in pixels.

Syntax

object.**Top** [=*value*]

Parameters

object

An expression that evaluates to the Toolset object.

value

A **Long** that sets the y-coordinate of the window's top edge in pixels.

Remarks

The **Top** property has the **Long** type.

The **Top** property gets or sets the distance in pixels between the top edge of the screen and the main Toolset window.

TreeView

Gets or sets whether the treeview is displayed.

Syntax

object. **TreeView** [=*boolean*]

Parameters

object

An expression that evaluates to a **Toolset** object.

boolean

A **Boolean** that sets whether the treeview is displayed. Possible values are:

- **True** - treeview is visible
- **False** - treeview is invisible

Remarks

The **TreeView** property has the **Boolean** type.

Not Yet Implemented.

Version

Gets the version number of CA Gen.

Syntax

object.**Version**

Parameters

object

An expression that evaluates to the Toolset object.

Remarks

The **Version** property has the **String** type (for example 6.0).

Visible

Gets or sets whether the CA Gen Toolset main window is visible.

Syntax

object.**Visible** [=*boolean*]

Parameters

object

An expression that evaluates to a **Toolset** object.

boolean

A **Boolean** that sets whether the window is visible. Possible values are:

- **True** - makes the window visible
- **False** - makes the window invisible

Remarks

The **Visible** property has the **Boolean** type.

Width

Gets or sets the width of a window in pixels.

Syntax

object.**Width** [=*value*]

Parameters

object

An expression that evaluates to the Toolset object.

value

A **Long** that sets the width of the window in pixels.

Remarks

The **Width** property has the **Long** type.

The **Width** property gets or sets the distance in pixels between the left and right edge of the main Toolset window.

WindowState

Gets or sets the state of a window.

Syntax

object.**WindowState** [=*state*]

Parameters

object

An expression that evaluates to the Toolset object.

state

An Enumeration of type **TS_WINDOWSTATE** that sets the state of the window. Possible values are:

TS_WINDOWSTATE_MAXIMIZED - Maximizes the window.

TS_WINDOWSTATE_MINIMIZED - Minimizes the window.

TS_WINDOWSTATE_NORMAL - Makes the window a MDI child window.

WorkPath

Gets or sets the work path.

Syntax

object. **WorkPath**

Parameters

object

An expression that evaluates to the Toolset object.

Remarks

The **WorkPath** property has the **String** type.

Methods

Quit

Quits the Toolset Application.

Syntax

object.**Quit**

Parameters

object

An expression that evaluates to the Toolset object.

Windows

Gets the Windows object.

Syntax

object.**Windows**

Parameters

object

An expression that evaluates to the Toolset object.

Remarks

The **Windows** property has the **Windows** type.

The **Windows** object is a collection object that represents all open windows associated with the Toolset object.

View Object

A **View** object is a generic view of a specific View object. The generic object contains methods and properties common to all View objects. The specific object contains methods and properties that represent a specific set of model functionality in the Toolset.

Not Yet Implemented.

Properties

Toolset

Gets the CA Gen Toolset object.

Syntax

object.**Toolset**

Parameters

object

An expression that evaluates to a **View** object.

Remarks

The **Toolset** property has the **Toolset** type.

The **Toolset** object represents the entire CA Gen Toolset. It provides access to other objects in the CA Gen object model and provides methods and properties that affect the entire Toolset.

Parent

Gets the parent object.

Syntax

object.**Parent**

Parameters

object

An expression that evaluates to a View object.

Remarks

The **Parent** property's type for the View object is the Toolset object.

Type

Gets the specific View type.

Syntax

object.**Type**

Parameters

object

An expression that evaluates to a **View** object.

Remarks

The View property has the type **dsViewType**.

DsViewType is a Long type. Possible values are:

DsOpenAPI - Open API

Views Collection

The **Views** object represents a collection of available View objects. A View represents a specific set of functionality to access and (or) change model data.

Not Yet Implemented.

Properties

_NewEnum

References objects in the **Views** collection.

Syntax

object.**_NewEnum**

Parameters

object

An expression that evaluates to a **Views** object.

Remarks

With Visual C++, you can browse a collection to find a particular item by using the **_NewEnum** property or the Item method. In Visual Basic, you do not need to use the **_NewEnum** property, because it is automatically used in the implementation of **For Each ... Next**.

Not Yet Implemented.

Toolset

Gets the CA Gen Toolset object.

Syntax

object.**Toolset**

Parameters

object

An expression that evaluates to a **Views** object.

Remarks

The **Toolset** property has the **Toolset** type.

The **Toolset** object represents the entire CA Gen Toolset. It provides access to other objects in the CA Gen object model and provides methods and properties that affect the entire Toolset.

Not Yet Implemented.

Count

Gets the number of Views in the collection.

Syntax

object.**Count**

Parameters

object

An expression that evaluates to a **Views** object.

Remarks

The **Count** property has the Long type.

Not Yet Implemented.

Parent

Gets the parent object.

Syntax

object.**Parent**

Parameters

object

An expression that evaluates to a Views object.

Remarks

The **Parent** property's type for the Views object is the Toolset object.

Not Yet Implemented.

Methods

Item

Gets a specified View from the collection.

Syntax

object.**Item** [*index*]

-or-

object index

Parameters

object

An expression that evaluates to a Views object.

Index

A Variant that is a **Long** or **String** representing a View.

If you specifiy a **Long**, the Item method fetches the object by its one-based index in the collection.

If you specify a **String**, it must be the View's name.

Not Yet Implemented.

Window Object

The **Window** object represents a multiple document interface (MDI) client window in which a model is being edited. The **Window** is also referred to as a Diagram.

Properties

Active

Gets or Sets whether the window is active.

Syntax

object.**Active** [=*boolean*]

Parameters

object

An expression that evaluates to a window object.

boolean

A **Boolean** that sets the state of the object. Possible values are:

- **True** - Activates the object.
- **False** - Deactivates the object.

Return Values

The Active property returns one of the following values:

- **True** - The object is active.
- **False** - The object is not active.

Remarks

The **Active** property has the **Boolean** type.

Toolset

Gets the CA Gen Toolset object.

Syntax

object.**Toolset**

Parameters

object

An expression that evaluates to a Window object.

Remarks

The **Toolset** property has the **Toolset** type.

Caption

Gets the caption of a window.

Syntax

object.**Caption**

Parameters

object

An expression that evaluates to a **Window** object.

Remarks

The **Caption** property has the **String** type.

Height

Gets or Sets the height of a window.

Syntax

object.**Height** [=*value*]

Parameters

object

An expression that evaluates to a **Window** object.

value

A **Long** that sets the height of the window in pixels.

Remarks

The **Height** property has the **Long** type.

Index

Gets where the window lies in the Z-order of the windows.

Syntax

object.**Index**

Parameters

object

An expression that evaluates to a **Window** object.

Remarks

The **Index** property has the **Long** type.

Windows in the collection accessed by the **Windows** property of the **Toolset** object are ordered by Z-Order. Accordingly, you can use the **Index** property to determine where a given window lies in the Z-Order of all MDI client windows.

Left

Gets or Sets the x-coordinate of the window's left edge in pixels.

Syntax

object.**Left** [=*value*]

Parameters

object

An expression that evaluates to a **Window** object.

value

A **Long** that sets the x-coordinate of the window's left edge in pixels.

Remarks

The **Left** property has the **Long** type.

Next

Gets the next window in the tab sequence.

Syntax

object.**Next**

Parameters

object

An expression that evaluates to a **Window** object.

Remarks

The **Next** property has the **Window** type.

Parent

Gets the parent object.

Syntax

object.**Parent**

Parameters

object

An expression that evaluates to a **Window** object.

Remarks

The **Parent** property's type for the Window object is the Document object.

Previous

Gets the previous window.

Syntax

object.**Previous**

Parameters

object

An expression that evaluates to a **Window** object.

Remarks

The **Previous** property has the **Window** type.

The previous window is the one you see when you press CTRL+SHIFT+TAB.

Top

Gets or Sets the y-coordinate of the windows top edge in pixels.

Syntax

object.**Top** [=*value*]

Parameters

object

An expression that evaluates to a Window object.

value

A **Long** that sets the y-coordinate of the window's top edge in pixels.

Remarks

The **Top** property has the **Long** type.

Width

Gets or Sets the width of a window.

Syntax

object.**Width** [=*value*]

Parameters

object

An expression that evaluates to a Window object.

value

A **Long** that sets the width of the window.

Remarks

The **Width** property has the **Long** type.

WindowState

Gets or Sets the window state.

Syntax

object.**WindowState** [=*state*]

Parameters

object

An expression that evaluates to a Window object.

state

A **Long** of type **DsWindowState** that sets the state of the window. Possible values are:

- **dsWindowStateMaximized** - Maximizes the window.
- **dsWindowStateMinimized** - Minimizes the window.
- **dsWindowStateNormal** - Makes the window a MDI child window.

Remarks

The **WindowState** property has the type **DsWindowState**.

Methods

Close

Closes windows.

Syntax

object.**Close** ([**SaveChanges**])

Parameters

object

An expression that evaluates to a Window object.

SaveChanges

(Optional) An enum of type **DsSaveChanges** that indicates whether to save changes to a document before closing it. Following are the possible values, which have the **Long** type:

dsSaveChangesYes - Saves all changes and does not prompt the user (the default).

dsSaveChangesNo - Does not save changes, and does not prompt the user.

dsSaveChangesPrompt - Prompts the user to save changes. If the user chooses not to save the changes, **Close** returns the enum **dsSaveStatus** with the value **dsSaveCanceled**; otherwise **Close** returns **dsSaveSucceeded**.

Return Values

The **Close** method returns the enum **dsSaveStatus**, which has one of the following values:

dsSaveCanceled - Indicates that the user chose not to save changes to the document.

dsSaveSucceeded - Indicates that the user saved changes to the document.

Windows Object

The **Windows** object represents the collection of open windows. Each window in the collection is represented by a Window object. A window is also commonly referred to as a diagram in the CA Gen terminology.

Properties

_NewEnum

References items in the collection

Syntax

object.**_NewEnum**

Parameters

Object

An expression that evaluates to a **Windows** object.

Remarks

With Visual C++, you can browse a collection to find a particular item by using the _NewEnum property or the Item method. In Visual Basic, you do not need to use the **_NewEnum** property because it is automatically used in the implementation of **For Each &ldots; Next**.

Toolset

Gets the CA Gen Toolset object.

Syntax

object.**Toolset**

Parameters

object

An expression that evaluates to a **Windows** object.

Remarks

The **Toolset** property has the **Toolset** type.

Count

Gets the number of windows.

Syntax

object.**Count**

Parameters

object

An expression that evaluates to a **Windows** object.

Remarks

The **Count** property has the **Long** type.

Parent

Gets the parent object.

Syntax

object.**Parent**

Parameters

object

An expression that evaluates to a Windows object.

Remarks

The **Parent** property's type for the Windows object is the Toolset object.

Methods

Arrange

Tiles, maximizes, or minimizes all open windows.

Syntax

object.**Arrange** *state*

Parameters

object

An expression that evaluates to a **Windows** object.

state

An enum of type **DsArrangeStyle** that sets the state of the windows. Following are the possible values, which have the **Long** type:

dsCascade - Cascades the windows.

dsMinimize - Minimizes the windows.

dsTileHorizontal - Tiles the windows horizontally.

dsTileVertically - Tiles the windows vertically.

CloseAll

Closes all open windows.

Syntax

object.**CloseAll** [*SaveChanges*]

Parameters

object

An expression that evaluates to a **Windows** object.

SaveChanges

(Optional) An enum of type **DsSaveChanges** that indicates whether to save changes to one or more documents before closing them. Following are the possible values, which have the **Long** type:

dsSaveChangesYes - Saves the changes (the default).

dsSaveChangesNo - Does not save the changes.

dsSaveChangesPrompt - Prompts the user to save changes. If the user chooses not to save the changes, **CloseAll** returns the enum **dsSaveStatus** with the value **dsSaveCanceled**; otherwise **CloseAll** returns **dsSaveSucceeded**.

Remarks

The **CloseAll** method returns the enum **dsSaveStatus**, which has one of the following values:

dsSaveCanceled - Indicates that the user chose not to save changes to the document.

dsSaveSucceeded - Indicates that the user saved changes to the document.

Item

Gets a specified window from the collection

Syntax

object.**Item** [*index*]

-or-

object index

Parameters

object

An expression that evaluates to a Windows object.

index

A **Variant** that is a **Long** or **String** representing the window.

If you specify a **Long**, the **Item** method fetches the object by its one-based index in the collection.

If you specify a **String**, it must be the window's caption.

Index

A

C

D

E

H

I

L

M

N

O

P

R

S

T

U

W