The CASE Interoperability Message Sets: *Release 1.0*

FINAL VERSION

1 October 1992

digital

SiliconGraphics
Computer Systems

SunSoft
A Sun Microsystems Company

The information contained in this document is in the public domain and is free for anyone. If you republish this document, or a portion of it, please, as a courtesy, reference this document as your source ("The CASE Interoperability Message Sets: Release 1.0" — published by Digital, SGI, and SunSoft (A Sun Microsystems Company), September 1992).

Digital and the Digital logo are registered trademarks of Digital Equipment Corporation.

SGI and the Silicon Graphics logo are registered trademarks of Silicon Graphics, Inc.

SunSoft and the SunSoft logo are registered trademarks of Sun Microsystems, Inc.

Table of Contents

Chapter 1: Introduction

CASE tool users require a high degree of tool integration to enhance developer productivity and to allow them to deliver applications more rapidly. While messaging technologies exist today that support easy, flexible "plug-and-play" tool integration, the lack of standard CASE message sets has inhibited integration.

Digital™, SGI™, and SunSoft™ offer this document as a significant step toward achieving more rapidly the goal of CASE message set standardization. This document contains the high-level semantic — not syntactic —specification of messages in sets. These messages are intended to be abstract and implementation independent so that CASE tools can maintain algorithmic consistency when integrating these messages on reasonably robust messaging technologies in different environments.

As systems vendors, Digital, SGI, and Sun Microsystems have gained extensive message integration experience through internal CASE tool development, and have also consulted extensively with a wide variety of independent CASE tool suppliers to develop a detailed understanding of their integration requirements. For example, SunSoft has worked with CASE tool vendors (including SGI and SunPro) through the CASE Interoperability Alliance; Digital and SGI have worked directly with numerous CASE tool vendors on message integration issues. In addition, an attempt was made to accommodate the broadest range of existing and emerging technologies, including broadcast, multicast, and OMG COBRA-compliant architectures. This document compiles the common understanding that Digital, SGI, and SunSoft have of software vendor and end-user messaging requirements.

Digital, SGI, and SunSoft offer this document to help further industry-wide work in CASE messaging. While Digital, SGI, and SunSoft feel that significant progress has been made, it is recognized that further work needs to be done to develop a consensus standard accepted across the industry. This document will be presented to the ANSI X3H6 Committee, an accredited standards body with the charter to develop messaging and other standards for the CASE marketplace.

While further consensus work needs to be done, Digital, SGI, and SunSoft feel quite strongly that this document already provides rich integration functionality that CASE tool suppliers can use today to deliver significant, tangible value to users. Overall, the CASE integration problem is complex. A complete, exhaustive message integration solution is still a future vision; however, an evaluation of the usage scenarios, message sets and man pages in this document will demonstrate clearly that certain significant integration benefits can be obtained without further delay.

By organizing messages into discrete sets, CASE vendors can choose to add product integration on a set-by-set basis. Tool vendors can chose the type of integration value to deliver to their users — and will be able to add incremental message set functionality to their products as their users require more extensive integration functionality. Pairs, or groups, of vendors can adopt the same message set(s) and can coordinate their release plans so that their products will have "plug and play" compatibility for specific kinds of integration. As a result, end-user purchasers will place higher value on these integrated tools.

The CASE Interoperability Message Set is an *open* specification with no royalty or license fees. It will be continuously revised and modified to meet the expanding needs of CASE developers and users. Send questions, comments, and requests for information to the CASE Messaging Alliance at *casemsg.ext@sun.com*.

Note – This alias is a forum for detailed technical interactions and is not a sales/marketing forum.

Chapter 2: What are the CASE Interoperability Message Sets?

To achieve control flow integration in an integrated CASE environment, applications need to support messages that enable inter-application control. The *CASE Interoperability Message Set*s are the common message sets that provide this functionality for CASE applications. A powerful messaging definition that benefits both developers and users of CASE applications, the CASE Interoperability Message Sets allow applications to easily interact with other CASE applications. The CASE Interoperability Message Sets define a standard set of CASE functionality available through control integration services.

Overview of the CASE Interoperability Message Sets

At a very high level, the primary objective of the CASE Interoperability Message Sets is to provide control flow integration in an integrated CASE environment. This control integration allows applications to interchange state information and to provide services to each other in a controlled and well-behaved manner.

There are eight CASE Interoperability Message Sets. Each of these sets describes an area of functionality in the CASE environment.

1. Analysis and Design
2. Browsing
3. Building
4. Debugging
5. Desktop Services
6. Documentation and Multimedia
7. Editing
8. Version Management

The Analysis and Design Message Set

The Analysis and Design Message set defines messages for functions that deal with the software architecture of an application. Typical functions for manipulation include structure chart editors, data flow diagrams, and flow charts.

The Browsing Message Set

The Browsing message set defines messages for functions that process queries on source code.

The Building Message Set

The Building message set defines an interface to build utilities. This message set includes the initiation of build processes at different granularities, error notifications, and recording of dependencies.

The Debugging Message Set

The Debugging message set defines a standard interface for a debug utility. This message set includes the common functions associated with debug utilities.

The Desktop Services Message Set

The Desktop service message set defines generic tool and file functions plus display management functions. These messages are usually accepted by all tools.

The Documentation and Multimedia Message Set

The Documentation and Multimedia message set defines documentation and media tool functions. This message set includes messages for displaying, editing, and translating the format of documentation and media data.

The Editing Message Set

The Editing messaging set defines a detailed collection of messages for text editing tools. These messages focus on the definition and manipulation of buffers and views and on text editing functions.

The Version Management Message Set

The Version Management message set defines messages for tools that manage version histories of software artifacts. This message set includes checkin, checkout, and other related version control functions.

Chapter 3: Abstract Messaging Service Environment

The abstract messages defined in this document assume a suitable messaging environment or capability. This environment or capability must accept and deliver messages between applications and must be able to track the status of message delivery.

Messaging Environment Requirements

This section describes the fundamental requirements for a messaging environment in order to support the abstract messages described in this document. The detailed definition of these requirements and mapping of these requirements to any particular implementation is outside the scope of this document.

The following capabilities must be provided within the messaging environment:

- The ability to deliver messages representing requests for functionality;
- The ability to deliver messages representing notifications of events;
- A mechanism or set of services for application message registration;
- A set of policies regarding delivery of messages;
- A mechanism or set of services to allow consistent delivery of a set of messages.

Message Delivery

Messages may be of two distinct types: Requests or Notifications.

Request messages are used to request specific functional services be performed. An indication of the success or failure of the functional service must be provided to the requestor, as well as providing any additional reply required by the definition of the message.

Note – Functions that may be time consuming will return success or failure to indicate status of the initiation of a function. When the function actually completes the function will return a notification indicating the success or failure of the function. An example of this class of message would be the build message.

Notification messages are used to provide notification to other applications in the CASE environment of events and state changes and do not require a response or affirmation of receipt.

The messaging environment must provide a reliable mechanism or set of services for delivering both these types of messages, as well as delivering the responses associated with the request messages.

Message Registration

The messaging environment must provide a mechanism or set of services to allow an application to define which messages it is capable of receiving. This registration capability should also provide the appropriate initialization services required for messaging to other applications in the messaging environment. Such services may cover the identification of the application to the environment, and the set up of communications between the environment and the application.

Policies for Message Delivery

The messaging environment needs to have well defined set of policies that control the delivery of messages. While this document can not specify the implementation policy it is an important issued to consider.

Consider the following algorithm for dispatching messages.

When a message is sent the messaging environment will first:

1. Consider all currently available recipients of messages;
2. Finding no acceptable receiver consider doing one of the following:
   a. Auto-starting an application on behalf of the user
   b. Send a failure message back to the requester, at this point the requester can make an intelligent decision as whether to wait or report failure back to the user.

The autostart capability enables an environment implementation to expand to fit the user's needs without user intervention.

Message addressing

The messaging environment should provide a mechanism or set of services to enable one application to assure consistent delivery of a set of messages to a particular application or set of applications. For example, this would enable an application to have all its edit requests serviced by a single editor instance, or enable all notifications to be sent to a select group of applications. Such a capability should be defined by the implementation, including the mechanism(s) to initiate, define the attributes of, and terminate the connection.

A possible implementation of this idea is to use a form of addressing that identifies a receiver by the function it performs. For example, an editor application would identify itself as "ascii text editor" so that any message that requests an ascii text editor function could be handled by the application. This form of addressing also enables different applications to supply the same functions.

Chapter 4: CASE User Scenarios

The CASE Interoperability Message Sets are designed to support a very diverse and robust user environment. User interfaces from simple ASCII terminals to state of the art windowed graphical user interfaces are taken into consideration. The message sets can be customized to suit the varied needs of applications; for example, to accommodate a single user on a multi-tasking machine; or to accommodate distributed computing environments of networked single-tasking commuters with multi-tasking servers.

A major goal of this facility is to make it easy for the tool developer to make his tool available in many different CASE environments. The result is that end-users have access to a greater number of applications in their chosen environment.

The CASE Interoperability Message Sets allow an application to integrate and control other applications without user intervention. This section illustrates several scenarios that show how the CASE Interoperability Message Sets might be used.

User Scenario: Fixing Bugs

This scenario steps through a complete cycle of how to fix a bug for a released application. It begins with receiving the bug report and describes the process required to fix the problem.

1. **Larry receives a bug report that indicates his application has a problem.**
2. **Larry invokes his CASE environment.**
   A CASE user interface is displayed. The functions Larry wants to perform are available in this CASE user interface.
3. **Larry writes a test case that duplicates the failure stated in the bug report.**
4. **Larry selects the debug function to run the application against the test case.**
   a. A Debug request is sent.

b. The messaging environment selects the debugging application. It does not find an instance of the application running in Larry's CASE environment, so it automatically starts a debugger.

c. The debugging application receives the request and loads the binary.

5. **Larry tests the code and reviews the debugging state in the debugging window.**
   He finds a function call that appears to be passing incorrect arguments.

6. **Larry selects the edit function to edit this code.**

   a. The debugging tool sends an edit request.

   b. The source editor receives a message to edit the specified source file.

7. **Larry wants to modify the source code, so he selects the checkout function.**

   a. The source code editor sends a checkout request.

   b. The source code editor receives the checkout notification, and changes the buffer state to modifiable.

8. **Larry edits the source code to fix the bug and selects the Build function to build the application.**

   a. The build request is sent.

   b. The build application receives the build request, and performs the build.

   c. When the build completes, the build application sends the BuildFinished notice.

   d. The debugger receives the BuildFinished notice and reloads the newly built application binary.

9. **Larry retests the application to confirm the bug fix works.**

10. **Larry exits the CASE environment.**

    a. Quit requests are sent to the source code editor, debugger, version manager, and builder applications.

Description of the CASE Interoperability Message Sets

Each of the CASE Interoperability message sets are composed of several request and notification messages. These messages can be used by any CASE tool to provide integration functionality in that area. Each set can be used separately or in combinations as applicable to your application's needs.

The Analysis and Design Message Set

The Analysis and Design message set provides an interface to functions that support software design methodologies.

Table 5 Design and Analysis Message Set

Requests	**Notifications**
AD_Attach_Object	AD_Code_Generated
AD_Check_Object	AD_Document_Generated
AD_Create_Object	AD_Edit_Centered
AD_Delete_Object	AD_Object_Attached
AD_Detach_Object	AD_Object_Checked
AD_Edit_Center	AD_Object_Created
AD_Export_Object	AD_Object_Deleted
AD_Generate_Code	AD_Object_Detached
AD_Generate_Document	AD_Object_Exported
AD_Load_Object	AD_Object_Imported
AD_Import_Object	AD_Object_Loaded
AD_Print_Object	AD_Object_Modified
AD_Rename_Object	AD_Object_Printed
AD_Save_Object	AD_Object_Renamed
	AD_Object_Saved

The following scenario represents a typical interaction.

- Brian is working on the design of a new application. He is using an object oriented methodology. On his screen are the design tools that make up his toolset.

1. **Brian invokes the design editor.**
   A blank window appears on his screen for him to create his design.
2. **Brian begins to creates objects and links them.**
   a. When an object is created, the design editor sends an AD_Object_Created notification; when an object is linked, the design editor sends an AD_Object_Attached message.
   b. The object editor receives each notification and logs it into the object database.
   c. Other interested tools also receive the notification messages and react to them.
3. **Brian selects useful objects from a menu and incorporates them into his design.**
   a. When an object is selected, the design editor sends an AD_Object_Import message to the design database.
   b. The design database receives the message and adds the object.
4. **Brian selects certain objects as "interfaces" to his design.**
   a. When an object is selected, the design editor sends an AD_Export_Object request to the object database.
   b. The object database accepts the request and marks the object in the object database.
5. **Brian completes his design.**
   a. The design editor sends requests to generate code and documentation.
   b. The code generator cannot generate the code and sends a AD_Code_Generated notification to the design editor to let it know the request failed.
6. **Brian selects an object to edit.**
   a. The design editor sends an Edit_center request to the object editor.
   b. The object editor displays the object that needs to be edited.

7. **Brian edits the object and retries the code and document generation.**
    a. The design editor sends requests to generate code and documentation.
    b. The code and documentation generator perform the requests and send a AD_Code_Generated and AD_Document_Generated notification, respectively, to the design editor.

The Browsing Message Set

The Browsing message set provides interface functions that support cross referencing, calling relationships and class hierarchy navigation.

Table 6 Browsing Message Set

Requests	**Notifications**
GetLinks	
GetObject	
DisplayLibrary	
DisplayLinks	
DisplayObjects	
FocusDisplay	
ModifyObjectDisplay	

The following is the scenario for the browser.

1. **Karen needs to fix some of the display code for her application. She needs to fix a routine called display-something. She is not sure of the name. From her editor she selects the find function and specifies the name "display" and the class routine.**
    a. The editor sends the GetObject message.
    b. The browser tool receives this message and searches its database for references to routines starting with the string "display" and returns the results to the editor.
    c. The editor sends the DisplayObjects message with the results returned back from GetObject.

d. The browser tool receives this message and displays references to the routine “displayResults”.

2. **Karen wants to see the callgraph for this routine so she selects the show-calls function, specifying called-by.**

   a. The GetLinks message is sent with the linkType of callsFrom.

   b. The browser tool receives this message and searches its database for all the routines called by “displayResults”. It passes back handles for all the calls.

   c. The editor receives the handles and then sends DisplayLinks message with these handles.

   d. The browser now displays a callgraph showing all the routines called by “displayResults”.

3. **Karen sees that “displayResults” calls the routine “gatherData”. She wants to see what routines are called from there. She selects the node in the browser and then selects the expand function.**

   a. The ModifyObjectDisplay message is now sent with the action expand. The browser receives this message and adds to the display all routines called by “gatherData”.

The Building Message Set

The Building message set defines the interface to a build subsystem. The build subsystem manages the processing and translation that recurs as source information is converted into a desired final product (such as an executable image).

There are two major functions the building message set performs:

1. Start a build process to create executable modules or other desired results.
2. Create and maintain dependency information that allows the build process to understand what steps need to be take to produce the desired results.

Table 7 Building Message Set

Requests	Notifications
Build	BuildFinished
CreateBuildTemplate	
StartBuild	
UpdateBuildTemplate	

The following scenario is typical of an interaction with a build tool.

1. **Richard creates and edits three files in a new area to begin the coding of a new project.**
2. **Richard selects the build template function in order to create the desired build rules automatically from the three source files.**
   a. The editor sends the CreateBuildTemplate message.
   b. The make based build subsystem scans the three source files and gen erates a makefile.
3. **Richard selects the Build target function in the editor.**
   a. The editor sends a build message to build the default target.
   b. The build subsystem performs the build, but fails due to an error in one of the source files.
   c. The build subsystem sends a BuildFinished notification.
4. **Richard corrects the error in the source file, and selects a test build function.**
   a. The editor sends a build message to compile the file just edited.
   b. The build subsystem successfully performs the compilation.
   c. The build subsystem sends a BuildFinished notification.
5. **Richard selects the build target function to complete the build, when he sees that his change compiled correctly.**
   a. The editor sends a build message to build the desired target.
   b. The build subsystem correctly completes the build.

c. The build subsystem sends a BuildFinished notification.

The Debugging Message Set

The Debugging message set provides a control interface for a debugging utility. The messages allow another tool to ask the debugging tool to load images, control execution, and control breakpoints. This kind of interface is used when close coupling is required between an editor, debugging tools, and editor toolset.

Table 8 Debugging Message Set

Requests	**Notifications**
Assign_DebugExpression	Debug_Executing
Debug	Debug_Exited
Delete_Trap	DebugContext_Set
Describe_DebugSymbol	Debug_Resumed
Eval_DebugExpression	Debug_Stopped
Execute_Debug	Debug_Unloaded
Load_Debug	Debug_Loaded
ResumeAt_Debug	Trap_Deleted
RunUntil_Debug	Trap_Hit
UnLoad_Debug	Trap_Set
Set_Trap	DebugScope_Set
Delete_Trap	Debug_Executing
Execute_Debug	DebugStopped
Terminate_Debug	Debug_Resumed
StepInto_Debug	Debug_Exited
StepOut_Debug	Debug_Unloaded
StepOver_Debug	
Stop_Debug	
Describe_DebugSymbol	
Assign_DebugExpression	

Table 8 Debugging Message Set *(Continued)*

Requests	Notifications
Eval_DebugExpression	
Get_DebugScope	
Set_DebugScope	

The following scenario illustrates a typical interaction.

▼ Kim is debugging a file. She has the debugging tool, a editor tool, and the build tools up on her display.

1. **Kim edits the file that she is debugging and commits her changes.**

   a. The editor makes the changes to the file.

2. **Kim requests a build.**

   b. The editor sends a *Build* request to the build tool.

   c. The build tool performs the build and, upon completion, sends a *BuildFinished* message back to the editor.

   d. The debugger sees the *BuildFinished* message and loads the new image.

3. **Kim marks a breakpoint on the editor window.**

   a. The editor sends a set_trap message to the debugger tool.

   b. The debugger sets the breakpoint.

4. **Kim clicks the Run button.**
   The application stops at the breakpoint.

The Desktop Services Set

The Desktop Services message set provides an interface for process control and display control of running tools. This kind of functionality provides control of integrated tool environments. These messages iconify, change the shape of, and display the layers of a tool's window. An application can also be requested to quit.

Table 9 Desktop Services Message Set

Requests	**Notifications**
Do_Command	Created
Get_Environment	Closed
Get_Geometry	Deleted
Get_Iconified	DeIconified
Get_Mapped	Iconified
Get_Status	Lowered
Quit	Mapped
Set_Environment	Modified
Set_Geometry	Moved
Set_Iconified	Opened
Set_Mapped	Raised
Signal	Reverted
Start	Saved
	Started
	Status
	Stopped
	UnMapped

The following scenario illustrates a typical interaction.

Astrid is browsing the source for an application. She is using a source browser tool to find her way through the complex file structure for this application.

1. **Astrid is editing a function and she finds an unfamiliar reference. She finds the definition of that reference in the source browser. Astrid requests an edit of the source code.**

a. The source browser sends an Edit message.

b. Since there is no editor present one is started.

c. The editor accepts the message and opens a window with the source code. When the editor is started it sends out a started notification.

2. Astrid views the code and closes the icon.

a. The editor sends an Iconify notice.

3. Astrid requests a different piece of source code.

a. The source browser sends out another Edit request.

b. The editor accepts it.

c. The source browser sends out a deiconify request.

4. Astrid brings the text editor to the top of her stack of windows.

a. The editor sends out a Raised mesage,

The Documentation and Multimedia Message Set

The Documentation and Multimedia message set provides the ability to invoke high-level editing and display services. You can use these facilities to create complex compound documents composed of text, video, sound, and other types of data. Applications for this kind of document include online help documents and advanced software documentation.

Table 10 Document and Media Exchange Message Set

Requests	Notifications
Abstract	
Deposit	
Display	
Edit	
Translate	

Typically, an icon that represents the media object is embedded in the document. Upon selection of an embedded object, a message is sent and the messaging environment invokes an appropriate external media application and the object is played as illustrated in the following scenario.

1. **Jackie opens a document that contains multimedia objects.**
    a. The window shows the document with several icons representing various media types (such as sound, video, and graphics).
2. **Jackie double-clicks on the sound icon recently placed in the document by her manager.**
    a. A Display request is sent.
    b. The messaging environment selects the sound application. It does not find an instance of the application running in Jackie's environment, so it starts it and the recording is played.

The Editing Message Set

The Editing message set provides tight control of the editor from outside applications. These messages are generally used in situations where coupling between the editor and another tool (or tools) is required. An example of this coupling is an editor and a debugger.

The Editing message set differs from the Documentation and Multimedia message set because the editing messages provide a lower level for buffered text manipulation.

Table 11 Editing Message Set

Requests	**Notifications**
Open	
Close	
Paste	
Copy	
Set_View	
Seek	

Table 11 Editing Message Set

Requests	Notifications
Goto	
Annotate	
Deannotate	

The following scenario illustrates a typical interaction.

▼ Ron is debugging a code module. He has an editor window up with the source code displayed. He also is running a debugger with a binary of the same source module loaded.

1. **Ron sets a breakpoint on all references to a variable.**
   a. The debugging tool registers this breakpoint.
   b. The debugging tool send a series of messages to the editor to find and annotate all references to the variable with a breakpoint symbol.
   c. The debugger also sends a Seek request to display the starting point of where it will run.
2. **Ron selects the Run function.**
   a. The debugging tool stops at the annotated breakpoint and sends another Seek to reposition the editor's display.
3. **Ron examines the program state with the debugging tool and clears the breakpoint.**
   a. The debugging tool clears the breakpoint and sends a clear_annotate request to the editor.

The Version Management Message Set

The Version Management message set defines the interface to a version control subsystem. The messages allow integrated CASE toolsets to check in and check out elements from a version control subsystem. In addition, the version control subsystem is expected to send notifications that allow the CASE environment to react and monitor activity in the version control subsystem. This functionality can be particularly important in multiple developers or smart toolset designs where the tool builder is required to make many separate tools coordinate actions on versioned elements.

Table 12 Version Management Message Set

Requests	Notifications
CancelCheckOut	CheckedIn
CheckIn	CheckedOut
CheckOut	CheckOutCanceled
Create	Merged
Delete	
Differences	
Fetch	
GetAttributes	
GetCheckOuts	
GetContexts	
Merge	
SetAttributes	

The following scenario illustrates a typical interaction.

1. **Mark invokes the editor on a file.**

   a. The editor sends a CheckOut message to the Version Control system.

   b. The Version Control subsystem accepts the request. It checks out the file and returns a successful status to the editor. It also sends a CheckedOut notification.

2. **Mark edits his file and, upon completion, clicks the Quit button.**

   c. The editor sends a CheckIn request to the Version Control system.

   d. The version control system accepts the request. It checks in the file and sends a CheckedIn notification.

Appendix A: The CASE Interoperability Message Sets Manpages

The following sections contain a description of each of the messages which constitute the CASE Interoperability Message Sets. Each message described in these sections contain the information described in Table 13

Table 13 CASE Interoperability Message Set Descriptions

Type of Information	**Description**
header	A single line that describes the message in the following format: *MsgName*(*MsgKind*) where *MsgName* is the name of the message and *MsgKind* is either Request or Notification.
name	A one-line description of the message.
description	An explanation of the operation (event) that the message requests (announces).
required arguments	The arguments that must always be included in the message. A particular argument is described in the following format: <mode> <vtype> <argument name> where *mode* is either in, out, or inout; *vtype* is a programmer-defined string that describes what kind of data a message argument contains; and *argument name* is the name of the argument. The messaging service uses vtypes to match sent message instances with registered message patterns. By convention, a vtype maps to a single, well-known data type.
optional arguments	Additional arguments which may be included in a message.
return values	Possible status values returned with the request response.

Data types

The following paragraphs describe the data types in the following man pages. A collection of semantic types have been defined for use in these messages. These semantic types define the information content of a data parameter, but they give no indication of specific data structures to be used to implement the semantic type. The following semantic types are used in this document:

bufferID	A type for uniquely identifying buffers.
boolean	A type of logical values.
datastream	An identifier for a stream of data. This may be a file name in some implementations, or an opaque handle in others. The entire text of the file might be included in the message in some case.
infoID	An identifier or handle that may be used to store implementation specific identifiers of objects of interest, like messages, tools, databases, etc.
integer	An integer
location	A namedvalues collection of data used to specify a location in a datastream. The names used in a location may include the following: line, column, word, section, chapter, function.
messageID	A type for uniquely identifying messages.
namedvalues	An list of namedvalue pairs consisting of name and values.
oneof	A collection of possible values. The possible values are listed by name. The list of names used may be extended in a particular implementation, but the names listed here should be used wherever possible. Depending on the implementation, these might be implemented as integer, strings, enumerations, or something else.
opaque	An identifier or handle that may be used to store implementation specific identifiers of objects of interest, like messages, tools, databases, etc. In any given implementation, each opaque type in this document may be implemented in different ways.

programlocation	A namedvalues collection of data used to specify a location within an executable image. The names used in programlocation may include the following: imageName, relativeAddress, absoluteAddress, ModuleName, lineNumber, routineName, ...
relativelocation	A namedvalues collection of data used to specify a relative location in a datasteam or buffer. The names used in relativelocation may include the following: deltaline, deltaColumn, deltaWord, ...
statuscode	A value indicating the status of a particular message.
string	A text string
viewID	A type for uniquely identifying views.

Context Variables

Applications that use these messages will also have access to an set of environment or context variables. These are retrieved and set with the Get_Environment and Set_Environment messages. These massages are designed to be compatible with the environment variables defined and used in the POSIX standards. In some implementations, however, a different mechanism may be used. The following environment variables are defined and used in this document. These will be the default values if they are not specified in a message.

ADRepository
: The handle that points to the analysis and design repository.

BuilderRules
: A handle that points to build rules data.

CrossRefLibrary
: The library that contains cross reference information.

DefaultBranch
: For version managers that allow checkout on a branch, DefaultBranch specifies the branch where checkouts and checkins will be done.

ExecutableImage
A handle that point to the executable image being developed.

VMLibrary
The location of the version managment library.

WorkingDataArea
The place where work will be done. In many system this is the working directory.

Appendix B:The Analysis and Design Notification and Request Messages

AD_Code_Generated(Notification)

States that code has been generated for a particular model.

Description

This message notifies all interested tools that source code has been generated for the diagram. A Modified messages should also be sent for each file modified by this operation.

Required Arguments

in opaque diagram
The name of the diagram or model.

in string diagramType
Used to distinguish between diagrams of the same name but of different types.

in string language
The source language of the object.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object

- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in opaque requestID

A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

AD_Object_Checked(Notification)

The object has been consistency checked.

Description

Notifies all interested tools that a consistency check has been performed. An entire repository may be checked in this manner.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in statuscode status
Status of operation.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

AD_Object_Modified(Notification)

The object has been modified.

Description

Notifies all registered applications that an object has been modified. You will also need to generate Modified notifications for any files that were also changed.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in statuscode status
Status of operation.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

AD_Object_Created(Notification)

A new object has been created. The object name is assigned by the tool.

Description

Notifies all registered applications that an object has been created. You will also need to generate Created notifications for any files that were also changed.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in statuscode status
Status of operation.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

AD_Object_Deleted(Notification)

An object has been deleted.

Description

Notifies all registered applications that an object has been deleted. You will also need to generate Modified notifications for any files that were also changed.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

AD_Object_Loaded(Notification)

An object has been loaded into the graphical editor.

Description

Notifies all registered applications that an object has been loaded.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope

in opaque editor
The handle associated with the editor that has a particular diagram or model loaded.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

AD_Object_Saved(Notification)

An object has been saved.

Description

Notifies all registered applications that an object has been saved. You will also need to generate Modified notifications for any files that were also changed.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

AD_Object_Printed(Notification)

An object has been printed.

Description

Notifies all registered applications that an object has been printed.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:.

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in string format
The format of the output.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

AD_Object_Attached(Notification)

An object has been attached to another.

Description

Notifies all registered applications that an object has been attached to another.

Required Arguments

in string connectionType
The type of connection used for connecting these two objects.

in opaque objectID1
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in opaque objectID2
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in statusCode status
Status of operation.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

AD_Object_Detached(Notification)

Two objects have been detached from each other.

Description

Notifies all registered applications that an object has been detached to another.

Required Arguments

in string connectionType
The type of connection.

in opaque objectID1
A unique identifier for the object which may consist of the following:.

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in opaque objectID2
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in statusCode status
Status of operation.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

AD_Document_Generated(Notification)

States that code has been generated for a particular model.

Description

This message notifies all interested tools that documentation has been generated for the diagram. A Modified messages should also be sent for each file modified by this operation.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in string documentType
The type of document that was generated.

in string format
The format of the document generated.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

AD_Edit_Centered(Notification)

Center diagram in named editor viewport.

Description

Notifies recipients that the diagram has been centered.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in opaque editor
The handle associated with the editor that has a particular diagram or model loaded.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

AD_Object_Imported(Notification)

An object has been imported.

Description

Notifies interested applications that an object has been loaded in the specified format.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in stream importFrom
Where the object is imported from.

in string format
The format of the object in the import location.

in statuscode status
Status of operation.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

AD_Object_Exported(Notification)

An object has been exported.

Description

Notifies interested applications that an object has been dumped to the specified format.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in stream exportTo
Where the object is exported to.

in string format
The format that the object should be created in.

in statuscode status
Status of operation.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

AD_Object_Renamed(Notification)

An object has been renamed.

Description

Notifies interested applications that an object has been renamed.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in string newName
The new name of the object.

in statuscode status
Status of operation.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

AD_Check_Object (Request)

Check named object for consistency.

Description

Check named object for consistency.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

AD_Create_Object (Request)

Create a new object.

Description

Create a new object.

Required Arguments

out opaque objectID

A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in opaque requestID

A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Optional Arguments

in opaque repository

The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

`AD_Delete_Object(Request)`

Delete an object.

Description

Requests that an object be deleted.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

`AD_Load_Object(Request)`

Load an object.

Description

Requests that an object be loaded into an editing viewport. Results in a AD_Object_Loaded notification with the editor name.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

`AD_Edit_Center(Request)`

Center diagram.

Description

Center diagram in named editor viewport.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in opaque editor
The handle associated with the editor that has a particular diagram or model loaded.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

AD_Save_Object(Request)

Save an object.

Description

Request a save of the the named object.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

AD_Generate_Code(Request)

Request code generation.

Description

Request code generation in the specificed language based on the diagram or model.

Required Arguments

in opaque diagram
The name of the diagram or model.

in string language
The source language of the object.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

inout datastream code
Description of output. The reason this is inout is left as an exercise to the reader.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

AD_Generate_Document (Request)

Request document generation.

Description

Request a document be generated from a list of object in the specified type.

Required Arguments

in opaque documentList
A list of object from which the document will be generated.

in string documentType
The type of document to generate.

in string format
The format of the document.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Optional Required Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

inout datastream document
Description of output.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

AD_Print_Object (Request)

Print an object.

Description

Request that the object be printed in the desired format.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in string format
The format of the output.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

AD_Attach_Object(Request)

Connect two objects together.

Description

Join two objects by the specific connection type.

Required Arguments

in string connectionType
The type of connection used for connecting these two objects.

in opaque objectID1
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in opaque objectID2
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

AD_Detach_Object(Request)

Detach two objects.

Description

Detach the specified objects.

Required Arguments

in opaque objectID1
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in opaque objectID2
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Optional Arguments

in opaque repository

The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

AD_Import_Object (Request)

Import an object.

Description

Requests that an object be loaded in the specified format.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in stream importFrom
Where the object is imported from.

in string format
The format of the object in the import location.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

AD_Export_Object (Request)

Export an object.

Description

Requests that an object be dumped to the specified format.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

out stream exportTo
Where the object is exported to.

in string format
The format that the object should be created in.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

AD_Rename_Object (Request)

Rename an object.

Description

Request that an object be renamed.

Required Arguments

in opaque objectID
A unique identifier for the object which may consist of the following:

- name - name of object
- class - an objects class (i.e. respository, model, diagram, etc.)
- type - an objects type (i.e. model index, data dictionary, data flow diagram, etc.)
- scope
- signature

in string newName
The new name of the object.

in opaque requestID
A unique identifier supplied by the request originator to facilitate the matching of a notification with a particular request.

Optional Arguments

in opaque repository
The name of the repository where the model is stored. The repository may be set with the Set_Environment message, in which case, this argument would not be needed.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Appendix C: The Browsing Notification and Request Messages

GetLinks(Request)

Request to get link information.

Description

The GetLinks message requests that information about what objects are connected to a specific object via a specified kind of relationship.

Required Arguments

in opaque objectIdentifier
    A handle to the object that the requester wants information about.

in oneof linkType
    Specifies the type of relationship that should be returned. The following values are possible values to this argument:

    callsTo
        all the routines that make calls to the specified routine

    callsFrom
        all the routines that are called from the specified routine

    subClass
        all the classes that are subclasses of the specified class

    superClass
        all the classes that are superclasses of the specified class

Optional Arguments

in opaque libraryIdentifier
A handle for the library which should be searched to determine the link information.

out opaque objectList
A list of handles of all the objects that are related to the specified object via the specified link.

out opaque linkList
A list of handles to all the links that match the criteria. The link handles must contain enough information so that a browsing tool can display the link and the objects at each end of the link. The format of this link information is implementation defined.

Note – ObjectList and/or LinkList must be specified in this request.

in namedvalues optionsList
A list of implementation specific options.

Return Values

NO_MATCH - No objects found

NO_OUTPUT - No output argument was specified the Set_Environment request.

GetObject(Request)

Request to find an object.

Description

The GetObject message requests that references to a named object be returned.

Required Arguments

in string nameFilter
A filter that tells the cross referencer what names should be included in the results.

in oneof nameMatchType
Specifies the type of the name filter. The following values are recognized:

Literal
A literal name was specified for the name filter

RegularExp
A regular expression was specified for the name filter

CaseInsensitive
A case-insensitive name was specified for the name filter

out opaque queryResults
This is a list of entries where each entry contains a list of the following information:

1. A handle to the object that was found. Note that we can get the name from this handle.
2. A handle to the object in which the object is referenced. The contents of this handle is implementation specific. It may contain a file specification or an identifier to an object in a repository.
3. Location information which tells a browser where in the object the object is referenced. This may include things like line and column information.
4. The usage of the reference. The following values are recognized: call-reference, read-reference, write-reference, declaration and definition.

5. The class. The following values are recognized: function, parameter, variable, type definition and label.

Optional Arguments

in opaque libraryIdentifier
A handle for the library which should be searched

in location containingObject
A handle to the object in which the object is located. The contents of this handle is implementation specific. The handle can be used for scoping information. An editor can be positioned on a particular reference to “i”. This tells the cross referencer which “i”.

in location containingObjectlocation
Location information which tells a browser where in the object the object was referenced. This may include things like line and column information.

in oneof class
The type of reference to find: function, parameter, variable, type definition. label and class.

in oneof usage
The type of usage to find: call-reference, read-reference, write-reference, declaration and definition.

in namedvalues optionsList
A list of implementation specific options.

Return Values

NO_MATCH - No objects found

ILLEGAL_NAME - The name filter contains illegal symbols

NO_LIBRARY - No cross reference library is set; set it with the Set_Environment request

DisplayLibrary(Request)

Request to display a graphical representation of all the information in the cross reference library.

Description

The DisplayLibrary message requests that a graphical representation of all the objects in the cross reference library be displayed.

Required Arguments

in oneof linkType
Specifies the type of relationships that should be displayed. The following values are possible values to this argument:

callsTo
all the routines that make calls to the specified routine

callsFrom
all the routines that are called from the specified routine

subClass
all the classes that are subclasses of the specified class

superClass
all the classes that are superclasses of the specified_class

Optional Arguments

in opaque libraryIdentifier
A handle for the library whose information is to be displayed.

in namedvalues optionsList
A list of implementation specific options.

Return Values

NO_MATCH - No objects found

NO_LIBRARY - No cross reference library is set; set it with the Set_Environment request

DisplayLinks(Request)

Request to display a set of links.

Description

The DisplayLinks message requests that a set of links be displayed for the user. It is up to the implementation to decide whether this display will replace any existing display.

Required Arguments

in opaque linkList
A list of handles to all the links that should be displayed. This list can be obtained by sending the GetLinks request. The format of this link information is implementation defined.

Optional Arguments

in opaque libraryIdentifier
A handle for the library in which the objects are stored.

in namedvalues optionsList
A list of implementation specific options.

Return Values

NO_LIBRARY - No cross reference library is set; set it with the Set_Environment request

DisplayObjects(Request)

Request to display a set of information about an object or set of objects.

Description

The DisplayObjects message requests that a set of information about an object be displayed for the user. It is up to the implementation to decide whether this display will replace any existing display. Note that this message can display information that is returned from the GetObjects message.

Required Arguments

in opaque objectInfo
This is a list of query entries where each entry contains a list of the following information:

1. A handle to the object that was found.
2. A handle to the object in which the object is referenced. The contents of this handle is implementation specific. It may contain a file specification or an identifier to an object in a repository.
3. Location information which tells a browser where in the object the object is referenced. This may include things like line and column information.
4. The usage of the reference. The following values are recognized: call-reference, read-reference, write-reference, declaration and definition.
5. The class. The following values are recognized: function, parameter, variable, type definition and label.

Optional Arguments

in opaque libraryIdentifier
A handle for the library in which the objects are stored.

in namedvalues optionsList
A list of implementation specific options.

Return Values

NO_LIBRARY - No cross reference library is set; set it with the Set_Environment request

FocusDisplay(Request)

Request to focus the display on a particular object.

Description

The FocusDisplay message requests that the display be focused on a particular object. This message is sent when a tool wants to indicate that a particular object has some special significance. It is up to the implementation to decide what to do with the request. It may choose to make the object visible in the display or it may choose to highlight the object.

Required Arguments

in opaque objectIdentifier
  A handle to the object that should be focused on

Optional Arguments

in opaque libraryIdentifier
  A handle for the library in which the cross reference information is stored.

in namedvalues optionsList
  A list of implementation specific options.

Return Values

NO_LIBRARY - No cross reference library is set; set it with the Set_Environment request

ModifyObjectDisplay(Request)

Request to modify the display of objects.

Description

The ModifyObjectDisplay message requests that the display of objects be modified in some way.

Required Arguments

in opaque objectList
A handle to the objects whose display should be modified.

in oneof action
Specifies the action that should be taken on the specified objects. The following values are possible values to this argument:

expand
all specified objects should be expanded one level by showing another level of links leaving the object. The links shown will be the same as in the previous display. If the display current shows that routines a, b and c are called by main, then sending the expand request with object a will then add to the display all the routines called by a.

expandAll
all specified objects should be recursively expanded by showing additional levels of links leaving the object. The links shown will be the same as in the previous display.

collapse
all specified objects should be collapsed by removing all links leaving the object.

zoom
The display will zoom in on the specified objects.

Optional Arguments

in opaque libraryIdentifier
A handle for the library in which the cross reference information is stored.

in namedvalues optionsList

A list of implementation specific options.

Return Values

NO_LIBRARY - No cross reference library is set; set it with setEnvironment request.

Appendix D:The Debugging Notification and Request Messages

Debug_Loaded(Notification)

Notification the debugger has loaded named file.

Description

The Debugged message notifies that a debugger has loaded a file.

Required Arguments

in string file
The executable file that has been loaded by the debugger.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

Optional Arguments

in int pid
The process ID of a running process that the debugger has attached to.

in string coreFile
The snapshot of an executed program at some moment in time.

Trap_Set(Notification)

The debugger has set a trap

Description

The debugger has set a trap.

Required Arguments

in string file
The executable file that is being set with a trap.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in oneof trapType
An operation that can occur at a particular portion of a programs address space.

- entry
- exit

The following refer to watchpoints:

- read
- write
- readwrite
- custom

in string action
An action for a debugger to take when a trap occurs.

- stop - debugger halts program as when a trap is encountered.
- trace - a historical representation of a location executed.
- do:<string in native debugger command language>

in programlocation trapSpec
For the pre-defined trapTypes, a trapSpec is a specification of a portion of a program’s address space, in one of the following forms:

0xNNNNN
lineNum[:offset]
symbol

in string condition
A string specifying an expression.

in opaque trapID
Identifier assigned to the trap.

Trap_Deleted(Notification)

The debugger has deleted a trap

Description

The debugger has deleted a trap.

Required Arguments

in string file
The source file that has had its trap deleted.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in oneof trapType
An operation that can occur at a particular portion of a programs address space.

- entry
- exit

The following refer to watchpoints:

- read
- write
- readwrite
- custom

in string action
An action for a debugger to take when a trap occurs.

- stop - debugger halts program as when a trap is encountered.
- trace - a historical representation of a location executed.
- do:<string in native debugger command language>

in programlocation trapSpec
For the pre-defined trapTypes, a trapSpec is a specification of a portion of a program's address space, in one of the following forms:

0xNNNNN
lineNum[:offset]
symbol

in string condition
A string specifying an expression.

in opaque trapID
Identifier assigned to a trap.

`Trap_Hit(Notification)`

The trap specified in trapSpec has occurred.

Description

The message should be sent it the trap specified in trapSpec has occurred and if a condition is specified and it is true.

Required Arguments

in string file
The source file in which the trap occurred.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in oneof trapType
An operation that can occur at a particular portion of a programs address space.

- entry
- exit

The following refer to watchpoints:

- read
- write
- readwrite
- custom

in string action
An action for a debugger to take when a trap occurs.

- stop - debugger halts program as when a trap is encountered.
- trace - a historical representation of a location executed.
- do:<string in native debugger command language>

in programlocation trapSpec
For the pre-defined trapTypes, a trapSpec is a specification of a portion of a program's address space, in one of the following forms:

0xNNNNN
lineNum[:offset]
symbol

in string condition
A string specifying an expression.

in opaque trapID
Identifier assigned to a trap.

DebugScope_Set(Notification)

The debugger scope has been set.

Description

The position in the stack has been set.

Required Arguments

in string file
The executable file being debugged that had its scope set.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in opaque frames
A top-first list of stack frames.

in string current
A zero-based index into frames.

Optional Arguments

None.

`Debug_Executing(Notification)`

Execution has been initiated.

Description

The target process of the debugger has entered a running state.

Required Arguments

in string in debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in string file
The executable file being debugged that is being run.

in string stdin
file, possibly empty.

in string stdout
file, possibly empty.

in string stderr
file, possibly empty.

Optional Arguments

in string parameters
A list of arguments that the program being debugged accepts.

Debug_Stopped(Notification)

Execution of the program being debugged has been stopped.

Description

The program has stopped due to termination, a stop request, or it has encountered a trap. The program is still in a loaded state and able to continue.

Required Arguments

in string file
The executable file being debugged that has been stopped.

string in debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in programlocation trapSpec
For the pre-defined trapTypes, a trapSpec is a specification of a portion of a program's address space, in one of the following forms:

0xNNNNN
lineNum[:offset]
symbol

Optional Arguments

None.

Debug_Resumed(Notification)

Execution has been resumed.

Description

The program has been in a stopped state and has resumed execution.

Required Arguments

string in file
The executable file that has been resumed.

string in debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in programlocation trapSpec
For the pre-defined trapTypes, a trapSpec is a specification of a portion of a program's address space, in one of the following forms: (In this case, the position from where the program resumed exection.)

0xNNNNN
lineNum[:offset]
symbol

Optional Arguments

None.

Debug_Exited(Notification)

The program being debugged has exited.

Description

The program has either normally completed or aborted due to some error. This implies that the program cannot resume or do step debugging.

Required Arguments

in string file
    The executable file that has exited.

in opaque debugClientID
    A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in statuscode exitStatus
    The integer exit status of a program that has exited while being debugged.

Optional Arguments

None.

Debug_Unloaded(Notification)

The debugger has unloaded the program being debugged.

Description

The debugger has unloaded the program being debugged.

Required Arguments

in string file
The executable file that has been unloaded.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

Optional Arguments

None.

`Load_Debug(Request)`

Ask that a file be loaded into a debugger.

Description

A request to the debugger to load the program named by the file and symbol information required to start the program. If the debugger is already has a file loaded, it will be replaced by the file specified in the file.

Required Arguments

in string file
The executable file that is to be debugged.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

Optional Arguments

in int pid
The running process. This assumes that the debugger has the ability to attach itself to a currently execution process.

in string coreFile
The image of the executing program.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Unload_Debug(Request)

Ask a debugger to unload the image it's debugging.

Description

The debugger will unload the image that is currently resident. It is up to the debugger to determine how to handle the different states of the debugging process.

Required Arguments

in string file
  The executable file that is to be debugged.

in opaque debugClientID
  A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Set_Trap(Request)

Set a trap.

Description

Cause execution to perform a predetermined action when trapSpec occurs (and optionally if condition is true), returning the ID of the trap created.

Required Arguments

in string file
The executable file that is being set with a trap.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in oneof trapType
An operation that can occur at a particular portion of a programs address space.

- entry
- exit

The following refer to watchpoints:

- read
- write
- readwrite
- custom

in string action
An action for a debugger to take when a trap occurs.

- stop - debugger halts program as when a trap is encountered.
- trace - a historical representation of a location executed.
- do:<string in native debugger command language>

in programlocation trapSpec
For the pre-defined trapTypes, a trapSpec is a specification of a portion of a program's address space, in one of the following forms:

0xNNNNN
lineNum[:offset]
symbol

out opaque trapID
Identifier assigned to a trap.

Optional Arguments

in string condition
A string specifying an expression.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Delete_Trap(Request)

Delete a trap.

Description

Delete the specified trap.

Required Arguments

in string file
The executable file to have its traps deleted.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

One of the following sets is required to positively identify the trap:

in opaque trapID
Identifier assigned to a type.

Or:

in oneof trapType
An operation that can occur at a particular portion of a programs address space.

- entry
- exit

The following refer to watchpoints:

- read
- write
- readwrite
- custom

in string action
An action for a debugger to take when a trap occurs.

- stop - debugger halts program as when a trap is encountered.

- trace - a historical representation of a location executed.
- do:<string in native debugger command language>

in programlocation trapSpec
For the pre-defined trapTypes, a trapSpec is a specification of a portion of a program's address space, in one of the following forms:

0xNNNNN
lineNum[:offset]
symbol

in string condition
A string specifying an expression.

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Execute_Debug(Request)

Start execution from beginning.

Description

Run the program from the beginning with the supplied arguments. The debugger is responsible for loading the necessary objects.

Required Arguments

in string file
The executable file that is to be debugged.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in string stdin
file for output, possibly empty.

in string stdout
file for output, possibly empty.

in string stderr
file for output, possibly empty.

Optional Arguments

in string parameters
A list of arguments that the program being debugged accepts.

Stop_Debug(Request)

Stop execution, returning the current execution location.

Description

The stopped execution of the program should allow the user to inspect characteristics of the program being debugged and to make changes if necessary. The debugger will fail the message if there is not a currently running program.

Required Arguments

in string file
The executable file that is to be stopped.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

out programlocation trapSpec
For the pre-defined trapTypes, a trapSpec is a specification of a portion of a program's address space, in one of the following forms:

0xNNNNN
lineNum[:offset]
symbol

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

`ResumeAt_Debug(Request)`

Resume execution, optionally at new location.

Description

The program should start from a desired location if specified, if not the program will start from the beginning. The debugger will fail the message if the program is not already stopped.

Required Arguments

in string file
    The executable file that is to be resumed.

in opaque debugClientID
    A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

Optional Arguments

in programlocation trapSpec
    For the pre-defined trapTypes, a trapSpec is a specification of a portion of a program's address space, in one of the following forms:

0xNNNNN
lineNum[:offset]
symbol

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

`RunUntil_Debug(Request)`

Run program until given point.

Description

The program should run until the specified eventSpec is encountered. The request performed should be on a stopped program.

Required Arguments

in string file
    The executable file that is to be run.

in opaque debugClientID
    A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in programlocation trapSpec
    For the pre-defined trapTypes, a trapSpec is a specification of a portion of a program's address space, in one of the following forms:

0xNNNNN
lineNum[:offset]
symbol

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

`Terminate_Debug(Request)`

Terminate the program being debugged.

Description

This reverts the program to newly loaded image resetting any conditions such as action, etc.

Required Arguments

in string file
: The executable file that is to be terminated.

in opaque debugClientID
: A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in string signal
: A specification that states the type of termination to be perform

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

`StepInto_Debug(Request)`

Execute one [count] statement[s], stepping into function calls, and stop, returning the current execution location.

Description

The steps executed should be performed on a stopped program that has been stopped by encountering a trap or receiving a stop message.

Required Arguments

in string file
The executable file that is to be stepped through.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

out programlocation trapSpec
For the pre-defined trapTypes, a trapSpec is a specification of a portion of a program's address space, in one of the following forms:

0xNNNNN
lineNum[:offset]
symbol

Optional Arguments

in int count
An integer amount specifying the number of statements.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

StepOut_Debug(Request)

Continue execution until the current scope is returned from, and reply with the current execution location.

Description

The current scope is normally defined by the target language, but usually represents the current function. The execution performed should be on a stopped program that has been stopped by encountering a event type or receiving a stop message from.

Required Arguments

in string file
The executable file that is to be executed.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

out programlocation trapSpec
For the pre-defined trapTypes, a trapSpec is a specification of a portion of a program's address space, in one of the following forms:

0xNNNNN
lineNum[:offset]
symbol

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

StepOver_Debug(Request)

Execute one [count] statement[s], stepping past function calls, and stop, returning the current execution location.

Description

The execution performed should be on a stopped program that has been stopped by encountering a trap or receiving a stop message.

Required Arguments

in string file
The executable file that is to be stepped through.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

out programlocation trapSpec
For the pre-defined trapTypes, a trapSpec is a specification of a portion of a program's address space, in one of the following forms:

0xNNNNN
lineNum[:offset]
symbol

Optional Arguments

in int count
An integer amount specifying the number of statements.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Describe_DebugSymbol(Request)

Provide description of identifier.

Description

The description would usually be the declaration of the given identifier or type. This is useful for getting members of a structure, union, or enumerated type.

Required Arguments

in stringfile
    The executable file with the symbol.

in opaque debugClientID
    A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in string symbol
    A string specifying a source-language identifier.

out string description
    A string containing a human-readable description.

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Assign_DebugExpression(Request)

Assign an expression to a specific varible.

Description

Assign the value of an expression to a variable.

Required Arguments

in string file
The executable file being debugged associated with the expression.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in string expression
A statement consisting of some combination of operands and operators.

in oneof language
A specificaton of the syntax of the expression such as follows:

- c
- pascal
- fortran
- special

in string target
A variable that is to receive the value of the expression.

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Eval_DebugExpression(Request)

Evaluate an expression and return the results.

Description

This request allows for the evaluation of an expression without the operation having side effects such as an assignment of a variable.

Warning – The expression should avoid assignment statements, which may defeat the purpose of the this request.

Required Arguments

in string file
The executable file that is associated with the expression.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in string expression
A statement consisting of some combination of operands and operators.

in oneof language
A specificaton of the syntax of the expression such as follows:

- c
- pascal
- fortran
- special

out string results
A numeric result from the evaluation of the expression.

Optional Arguments

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

`Get_DebugScope(Request)`

Return the stack and the current frame therein.

Description

This provides location information of the stack of the file being debugged.

Required Arguments

in string file
The executable file being debugged from which to receive the context.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

out opaque frames
A top-first list of stack frames.

out int current
A zero-based index into `frames`.

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

`Set_DebugScope(Request)`

Set the current stack frame.

Description

Allows setting the location of the stack of the file being debugged.

Required Arguments

in string file
The executable file being debugged that is to have its context set.

in opaque debugClientID
A value specifying the identification of the debugging process. This is to be used as a method of allowing the process to keep track of which debugger it is working in behalf of.

in int current
A zero-based index into `frames`.

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Appendix E:The Building Notification and Request Messages

StartBuild(Request)

Description

A build is started. A number of related functions are possible, depending on the target parameter used.

Required Arguments

in opaque target

in opaque source
Either the target or the source parameter is required, but not both. Both, however, may be supplied together. The target of the build operation is a handle or id of the thing to build. The target handle may describe a desired executable image, a target name in a makefile, or objects in a repository. The source parameter represents a source file that may have recently been changed and that needs to be recompiled. If the source is present, it will be precessed in an implementation specific manner; if the target is also present, it will be generated, using the specified source. It is up to the implementation to determine how to interpret these handles.

Optional Arguments

in opaque ruleset
Rules is a handle to an object that defines the build rules for the target to be built. The target handle may describe a makefile or a rule object in a repository. It is up to the implementation to determine how to interpret this handle. The type and the description of the target parameter varies, depending on which function is desired. The rules argument is optional. When omitted a default or a previously selected collection of build rules will be used.

in named

values options
The options are any necessary options needed to specify the desired build.

Returned Values

The Build request message returns a status value. It may have the following values:

Success - The build was started successfully.

NoTarget - No appropriate target was found.

NoRules - The specifed rules were not found, or no default rules exist.

BadFile - A requested source file can not be found.

GenerateBuildRules(Request)

Description

The GenerateBuildRules message requests that build rules be created or updated, based on the contents of source files in sourcelist. The most commonly used form of build rules is a makefile.

Optional Arguments

inout opaque ruleset

The name of the set of build rules to generate or update. If the named rules do not exist they are created; if they exist, they are updated. If this optional ruleset parameter is omitted, the build application will use the value of the BuildRules context variable, or, if BuildRules is undefined, its own default rules name.

in namedvalues options

A list of more options that may be needed to generate or update the build rules.

in opaque sourcelist

The sourcelist contains a list of handles to things that should be examined while creating the build rules. The handles may be for individual source files, directories containing source files, or for the names of objects or set of objects in a repository.

Interpretation of these handles is up to each implementation.

Returned Values

The GenerateBuildRules request message returns a status value. It may have the following values:

Success - The desired template was successfully created.

BadOption - An option was invalid.

BadRules - The specifed build template was invalid.

BadSourceList - The sourcelist contained one or more invalid element.

`BuildFinished(Notification)`

Description

The BuildFinished, message may be sent by a build tool to notify other tools that a build procedure has completed. The message is sent whether or not errors are discovered. Information about errors that might have occured during the build are not sent in this message.

Required Arguments

in opaque target
    The target of the BuildFinished operation is a handle or id of the thing that was built (or at least attempted to be built)in datastream buildlog

    A handle to all of the diagnostic messages generated during the build process.

in oneof status
    Specifies the result or exit status of the build procedure completion: Success, and Failure are possible values.

in opaque ruleset
    Ruleset is a handle to an object that defines the build rules for the target that was built. The target handle may describe a makefile or a rule object in a repository.

in namedvalues options
    The options are any build options that were used for the build that just completed.

Optional Arguments

in opaque source
    The source is the source file supplied in the StartBuild message, if one was supplied.

Appendix F: The Desktop Services Notification and Request Messages

`Created/Deleted(Notification)`

Notification that entities (for example, files) have been created or deleted.

Description

The Created/Deleted message notifies interested tools whenever a tool creates or deletes one or more entities.

Required Arguments:

in infoID ID
The identifier of the created or deleted entity. If more than one entity is created or deleted in the same logical event, extra ID arguments may be present.

Do_Command(Request)

Requests in a tool's native command language that a command be performed.

Description

The Do_Command message requests that the receiving tool perform a command. The request is stated in the receiving tool's native command language.

When the request includes the optional *msgID* argument, the handler can send an immediate point-to-point status notification back to the requesting tool if the requested operation is expected to require an extended amount of time.

Required Arguments

in string command
The command being requested to be performed.

out string results
The results of the completed command. The results are returned as if the command had been executed locally to the requesting tool.

Optional Arguments

in infoID msgID
A unique identifier for this message.

Get_Status(Request)

Requests that a tool's current status be returned.

Description

The Get_Status message retrieves either the current status of a tool or the current status of a specific operation that is being performed by a tool.

Required Arguments

inout statusCode status
The status to be retrieved.

inout string vendor
The name of the vendor of the receiving tool.

inout string toolName
The name of the receiving tool.

inout string toolVersion
The version of the receiving tool.

Optional Arguments

in infoID msgID
A unique identifier for this message.

Iconified/DeIconified(Notification)

Notification that a tool has iconified or de-iconified.

Description

The Iconified/Deiconified message notifies interested tools whenever a tool iconifies or de-iconifies the window(s) .

Required Arguments

in string vendor
The name of the vendor of the iconified or de-iconified tool.

in string toolName
The name of the iconified or de-iconified tool.

in string toolVersion
The version of the iconified or de-iconified tool.

Optional Arguments

in infoID msgID
A unique identifier for this message.

in infoID buffID
The identifier of any buffer associated with the window(s) that were iconified or de-iconified.

Mapped/UnMapped(Notification)

Notification that a tool has mapped or unmapped its window(s) to or from the screen.

Description

The Mapped/UnMapped message notifies interested tools whenever a tool maps or unmaps the window(s).

Required Arguments

in string vendor
The name of the vendor of the mapped or unmapped tool.

in string toolName
The name of the mapped or unmapped tool.

in string toolVersion
The version of the mapped or unmapped tool.

Optional Arguments

in messageID msgID
A unique identifier for this message.

in infoID buffID
The identifier of any buffer associated with the window(s) that were mapped or unmapped.

Modified(Notification)

Notification that an entity (for example, a file) has been either modified or reverted to its prior state.

Description

The Modified message notifies interested tools whenever a tool first makes changes to a volatile, non-shared (for example, in-memory) representation of the persistent state of an entity (such as a file). The Reverted message notifies interested tools whenever a tool discards the modifications made to a volatile, non-shared (for example, in-memory) representation of the persistent state of an entity (such as a file).

Required Arguments

in infoID ID
    The identifier of the modified or reverted entity.

Moved(Notification)

Notification that an entity (for example, a file) has been moved.

Description

The Moved message notifies interested tools whenever a tool changes the location of a persistent entity.

Required Arguments

in string oldPath
The path from which the file has been renamed. If *oldPath* is a relative path, then it is relative to the directory that contains the path to which the file has been renamed.

in infoID oldID
in infoID newID
The old and new identifiers of the moved entity.

Opened/Closed(Notification)

Notification that an entity (for example, a file) has been opened or closed.

Description

The Opened/closed message notifies interested tools whenever a tool opens or closes a volatile, non-shared (for example, in-memory) representation of a persistent entity (such as a file).

Required Arguments

in infoID ID
The identifier of the opened or closed entity.

Quit (Request)

Requests that an operation, or an entire tool, terminate.

Description

The Quit message requests that the receiving tool terminate either an operation it is performing, or itself.

- If the optional *theRequest* argument is included, this message requests that the receiving tool terminate the indicated operation.

Note – This request fails when the termination is not performed; for example, because the *silent* argument is false, allowing the user to cancel the quit command.

Required Arguments

in boolean silent
The boolean value that indicates whether the receiving tool is allowed to request user input before it terminates.

in boolean force
The boolean value that indicates whether the receiving tool can terminate although it would normally not terminate under the current circumstances.

For example, a tool that does not quit with unsaved changes unless it asks the user whether the changes are to be saved will terminate if this argument is true although the data will be lost because user has not been asked whether changes are to be saved.

Optional Arguments

in messageID theRequest
The operation that is to be terminated.

Note – To be able to terminate a request, the requesting tool that initiated the operation to be terminated must have included an optional messageID in the requesting message; and the handler must have identified itself to the requesting tool by returning a Status notification to it.

Raised/Lowered(Notification)

Notification that the stacking order of a tool's window(s) have changed.

Description

The Raised/Lowered message notifies interested tools whenever a tool raises or lowers the window(s) associated with the sending tool's procID.

Required Arguments

in string vendor
The name of the vendor of the raised or lowered tool.

in string toolName
The name of the raised or lowered tool.

in string toolVersion
The version of the raised or lowered tool.

Optional Arguments

in messageID msgID
A unique identifier for this message.

in infoID buffID
The identifier of any buffer associated with the window(s) that were raised or lowered.

`Saved(Notification)`

Notification that an entity (such as a file) has been saved to persistent storage.

Description

The Saved message notifies interested tools whenever a tool saves an entity (such as a file) to persistent storage.

Required Arguments

in infoID ID
The identifier of the saved entity.

Optional Arguments

none

Set_Environment/Get_Environment(Request)

Requests that a tool's environment either be set or retrieved.

Description

The Set_Environment/Get_Environment message requests that the value of the indicated environment variable(s) either be replaced or reported. This function may or may not be implemented using Unix environment variables.

Required Arguments

in string variable
The name of the environment variable to be set or retrieved.

inout string value
The value of the environment variable to be set or retrieved.

- If this argument is omitted for the Set_Environment request, the variable is removed from the environment. It is not considered an error if the specified variable does not exist.
- If this argument does not contain a value when used in the Get_Environment request, the variable was not present in the receiving tool's environment. This condition is not considered an error.

Optional Arguments

in string variable/inout string value
Extra pairs of these arguments may included.

in string cname

The name of the set of context definitions to be searched (or changed). If this parameter is omitted, an implementation defined default set of context parameters will be used. Some implementations may have only one defined set of context information.

Set_Geometry/Get_Geometry(Request)

Requests that a tool's on-screen geometry either be set or retrieved.

Description

The Set_Geometry message requests that the value of the on-screen geometry of the optionally specified window, or the value of the on-screen geometry of the window primarily associated with the receiving tool's procID if no window is specified, be either set or retrieved.

Required Arguments

in/out int width
in/out int height
in/out int xOffset
in/out int yOffset
The integer geometry values in pixels.

The return values for the Get_Geometry request are the actual new values, not the requested new values.

Optional Arguments

in messageID msgID
A unique identifier for this message.

in infoID buffID
The identifier of any buffer associated with the window for which the value is to be set or retrieved.

Set_Iconified/Get_Iconified(Request)

Requests that a tool's iconic state be set or retrieved.

Description

The Set_Iconified message requests that the value of the iconic state of the optionally specified window, or the iconic state of the window primarily associated with the receiving tool's procID if no window is specified, be either set or retrieved.

Required Arguments

in/out boolean iconic
The boolean value that indicates whether the specified window is iconified.

Optional Arguments

in messageID msgID
A unique identifier for this message.

in infoID buffID
The identifier of any buffer associated with the window for which the value is to be set or retrieved.

Set_Mapped/Get_Mapped(Request)

Requests that a tool's mapping to the screen be set or retrieved.

Description

The Set_Mapped/Get_Mapped message requests that value of the mapped state of the optionally specified window, or the mapped state of the window primarily associated with the receiving tool's procID if no window is specified, be either set or retrieved.

Required Arguments

in/out boolean mapped
The boolean value that indicates whether the specified window is mapped to the screen.

Optional Arguments

in messageID msgID
A unique identifier for this message.

in infoID buffID
The identifier of any buffer associated with the window for which the value is to be set or retrieved.

`Signal(Request)`

Requests that a (POSIX-style) signal be sent to a tool.

Description

The Signal message requests that the receiving tool send the indicated signal to itself.

Required Arguments

in integer theSignal
    The signal to be sent.

`Started/Stopped(Notification)`

Notification that a tool has started or terminated.

Description

The Started/Stopped message notifies interested tools whenever a tool starts or terminates.

Required Arguments

in string vendor
    The name of the vendor of the started or terminated tool.

in string toolName
    The name of the started or terminated tool.

in string toolVersion
    The version of the started or terminated tool.

`Status(Notification)`

Notification that a tool has status information to announce.

Description

The Status message notifies interested tools of a tool's general status information.

Required Arguments

in statusCode status
The status which is being announced.

in string vendor
The name of the vendor of the tool whose status is being announced.

in string toolName
The name of the tool whose status is being announced.

in string toolVersion
The version of the tool whose status is being announced.

Optional Arguments

in messageID msgID
A unique identifier for this message.

Appendix G: The Document and Multimedia Notifications and Requests

Abstract(Request)

Requests a summary representation of a document.

Description

The Abstract message requests that a summary representation of a document (for example, an icon or a video frame raster) be returned.

Required Arguments

in oneof mediaType
The name of a media format as indicated. The mediaType of a document allows messages about that document to dispatched to the right editor.

- PostScript
- text
- c-source
- ....

in datastream contents
The contents of the document.

out oneof mediaTypeOut
The name of a media format for output as indicated.

- PostScript
- text
- c-source
- ....

out opaque output
The abstracted document.

Optional Arguments

in messageID msgID
A unique string created by the message sender. The sending application includes this argument if it anticipates a need to communicate with the handler about this request before the request is completed; for example, you could include this argument to cancel the request.

inout interger size
The size of the abstraction.

Returned Values

`EDITOR_ERR_SIZE`
requested size is too small.

- On input, the region size is the maximum size of the abstraction. The returned abstraction must not exceed this size.
- On output, the region size is the actual size of the abstraction that is being returned; if the error `EDITOR_ERR_SIZE` is returned, the indicated region size is the smallest possible size the recipient is capable of returning.

Deposit(Request)

Requests that a buffer of shared interest be saved to its storage.

Description

The Deposit request the receiver to store a buffer of shared interest.

Required Arguments

in infoID ID
An editor-assigned identifier for a volatile, non-shared representation (for example, in-memory) of a document. The ID is either the identifier returned when this buffer was created by an Open request or the messageID of the Edit request that created this buffer.

Optional Arguments

inout datastream place
Describe location to save buffer to.

`Display(Request)`

Displays a document.

Description

The Display message requests that a document be displayed. Display is a generic term for the media the player performs; for example, an audiotool displays sound. The Display request invokes the requested playback mechanism (such as a video tool, an audio tool, or a text-to-speech synthesizer).

Required Arguments

in oneof mediaType
The name of a media format as indicated. The mediaType of a document allows messages about that document to dispatched to the right editor.

- PostScript
- text
- c-source
- ....

in datastream contents
The contents of the document.

Optional Arguments

in messageID msgID
A unique string created by the message sender. The sending application includes this argument if it anticipates a need to communicate with the handler about this request before the request is completed; for example, you could include this argument to cancel the request.

in string title
The name of the document.

`Edit(Request)`

Edits or composes a document.

Description

The Edit message requests that a document be edited.

Required Arguments

inout oneof mediaType
The name of a media format as indicated. The mediaType of a document allows messages about that document to dispatched to the right editor.

- PostScript
- text
- c-source
- ....

Optional Arguments

in messageID msgID
A unique string created by the message sender. The sending application includes this argument if it anticipates a need to communicate with the handler about this request before the request is completed; for example, you could include this argument to cancel the request.

in string title
The name of the document.

inout datastream contents
The contents of the document. If not supplied a new document will be created.

Translate(Request)

Translates a document from one media type to another media type.

Description

The Translate message requests that a document be translated from one media type to another media type and that a reply containing the translation be returned.

Required Arguments

in oneof mediaType
The name of a media format as indicated. The mediaType of a document allows messages about that document to dispatched to the right editor.

- PostScript
- text
- c-source
- ....

in datastream contents
The contents of the document.

out oneof mediaTypeOut
The name of a media format for output as indicated.

- PostScript
- text
- c-source
- ....

out opaque output
The translated document.

Optional Arguments

in messageID msgID

A unique string created by the message sender. The sending application includes this argument if it anticipates a need to communicate with the handler about this request before the request is completed; for example, you could include this argument to cancel the request.

Appendix H:The Editing Requests

Open(Request)

Open the buffer and return the handle.

Description

The Open request is used instead of the Document and Multimedia message, Edit when programmatic control over the editing process is desired.

Required Arguments

in oneof mediaType
The name of a media format as indicated. The mediaType of a document allows messages about that document to dispatched to the right editor.

- PostScript
- text
- c-source
- ....

in datastream contents
The data to be viewed or edited.

Only one of the following two arguments is required:

out bufferID docBuf
The identification of a buffer that holds the contents.

Or:

out viewID docView
The identification of a view of the buffer contents that relates to a portion of a viewable screen from a given location in the buffer.

in boolean readOnly
A boolean value indicating whether or not the buffer is modifiable.

Optional Arguments

in boolean mapped
Boolean value indicating whether the receiver should transition from a mapped to unmapped state as a consequence of this request. (This allows the sender to, e.g., set the screen geometry of the handler before mapping it, thus keeping the handler from blinking.)

in oneof shareLevel
newbuffer - The handler should create a new buffer. (This flag is not applicable when returning a bufferID.)

unmodbuffer - The handler may refuse an existing, unmodified buffer.

modbuffer - The handler may refuse an existing, modified buffer.

in location initialPos
A location in a document. Consists of one or more component and component identifier pairs.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Paste(Request)

Paste data to a buffer.

Description

Paste a desired selection to a named buffer. Use current point to paste.

Required Arguments

in bufferID docBuf
The identification of a buffer.

in oneof mediaType
The name of a media format as indicated. The mediaType of a document allows messages about that document to dispatched to the right editor.

- PostScript
- text
- c-source
- ....

in datastream contents
The data to paste.

in relativelocation offset
The value used to express the end point of the paste in reference to the insertion point. Any value used other than zero would result in a replacement of the existing data from the curent poing to the offset. Zero implies an insertion.

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Note – The use of the Paste request should not be thought of as the recommended implementation of selections if other mechanisms are available.

`Copy(Request)`

Copy contents from buffer.

Description

Copy a desired selection from a named buffer. Use current point to copy.

Required Arguments

in bufferID docBuf
The identification of a buffer.

out oneof mediaType
The name of a media format as indicated. The mediaType of a document allows messages about that document to dispatched to the right editor.

- PostScript
- text
- c-source
- ....

out datastream contents
The desired selection.

in relativelocation offset
The value used to express the location where the copy should end in reference to the insertion point.

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Note – The use of the Copy request should not be thought of as the recommended implementation of selections if other mechanisms are available.

`Set_View(Request)`

Create a view into the indicated buffer.

Description

Create a view into the indicated buffer.

Required Arguments

in oneof mediaType
The name of a media format as indicated. The mediaType of a document allows messages about that document to dispatched to the right editor.

- PostScript
- text
- c-source
- ....

in datastream contents
The data to be viewed.

in viewID docView
The identification of a view of the buffer contents that relates to a portion of a viewable screen from a given location in the buffer. The value of docView should be obtained by a previous Open request.

in boolean readOnly
A boolean value indicating whether or not the buffer is modifiable.

Optional Arguments

in oneof shareLevel

newbuffer - The handler should create a new buffer. (This flag is not applicable when returning a bufferID.)

unmodbufer - The handler may refuse an existing, unmodified buffer.

modbuffer - The handler may reuse an existing, modified buffer.

in location initialPos

A location in a document. Consists of one or more component and component identifier pairs.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

`Seek(Request)`

Seek to a position in a buffer.

Description

Request receiver to go to a position specified by a relative offset to the current point. If no offset is specified the location of the pointer is set to the beginning of the buffer.

Required Arguments

Only one of the following two arguments is required:

in bufferID docBuf
The identification of a buffer.

Or:

in viewID docView
The identification of a view of the buffer contents that relates to a portion of a viewable screen from a given location in the buffer.

in relativelocation offset
The value relative to the insertion point which the pointer will be moved.

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

`Goto(Request)`

Go to indicated position in buffer.

Description

Requests receiver to go to an absolute position in relation to the current point.

Required Arguments

Only one of the following two arguments is required:

in bufferID docBuf
The identification of a buffer that holds the contents.

Or:

in viewID docView
The identification of a view of the buffer contents that relates to a portion of a viewable screen from a given location in the buffer.

in location destination
A specification of the location to go to.

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

`Close(Request)`

Close the indicated buffer.

Description

Request receiver to close the indicated buffer or view.

Required Arguments

Only one of the following two arguments is required:

in bufferID docBuf
The identification of a buffer that holds the contents.

Or:

in viewID docView
The identification of a view of the buffer contents that relates to a portion of a viewable screen from a given location in the buffer.

in boolean inquisitive
A boolean value used to determine if the application should inquire the user before closing the buffer.

in boolean force
Determines whether unsaved changes to the buffer should cause the request to fail.

Optional Arguments

None.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Annotate(Request)

Display an annotation in (a view on) a buffer.

Description

Display an annotation in (a view on) a buffer.

Required Arguments

in viewID docView
The identification of a buffer that is to be annotated.

in location start
The start point for the annotation.

in relatinvelocation extent
A value representing the offset from the start point.

in string annotation
The annotation to set. Such as:

error

warning

breakpoint

breakpoint (inactive)

current PC

pending PC

custom

Optional Arguments

Extra sets of the start, extent, and annotation arguments may be included.

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Deannotate(Request)

Remove an annotation from (a view on) a buffer.

Description

Remove an annotation from (a view on) a buffer. If the optional arguments are not present, then the annotation is removed from all views on the buffer.

Required Arguments

in viewID docView
The identification of a view of the buffer contents that relates to a portion of a viewable screen from a given location in the buffer.

Optional Arguments

Multiple sets of the following three arguments may be used but all three must be present in any given set.

in location start
The start point for the annotation.

in relativelocation extent
A value representing the offset from the start point.

in string annotation
The annotation to set. Such as:

error

warning

breakpoint

breakpoint (inactive)

current PC

pending PC

custom

Return Values

Success - this operation was performed sucessfully

Failure - this operation failed, [detailed error messages will be defined]

Appendix I: The Version Management Notifications and Requests

CheckedOut (Notification)

Notification that a check out operation occurred

Description

The CheckedOut message notifies interested tools that a checkout operation occurred.

Required Arguments

in opaque objectIdentifier
A handle for the object that was checked out.

Optional Arguments

in opaque libraryIdentifier
A handle for the library from which the object was checked out.

in string version
The version of the module checked out.

in string branchName
The name of the branch on which the object was checked out. Note that not all version managers handle specifying a branch on check out.

in string comment
A string indicating the purpose of the modification

in namedvalues optionslist
A list of implementation specific options.

`CheckedIn(Notification)`

Notification that an object was checked in.

Description

The CheckedIn message notifies interested tools that checkin occurred.

Required Arguments

in opaque objectIdentifier
A handle for the object that was checked in.

Optional Arguments

in opaque libraryIdentifier
A handle for the library to which the object was checked in

in string comment
A string indicating how the object was modified.

in string version
The version of the module checked in.

in string branchName
The name of the branch on which the object was checked in. Note that not all version managers handle specifying a branch on checkin.

in namedvalues optionslist
A list of implementation specific options.

CheckOutCanceled(Notification)

Notification that a check out operation was canceled

Description

The CheckOutCanceled message notifies interested tools that check out was canceled.

Required Arguments

in opaque objectIdentifier
A handle for the object whose checkout was canceled

Optional Arguments

in string comment
A string indicating why the checkout was canceled

in namedvalues optionslist
A list of implementation specific options.

Merged(Notification)

Notification that some component versions have been merged into a new component version.

Description

The Merged message notifies interested tools that some component versions have been merged. A new component version has been created out of these versions.

Required Arguments

in opaque objectIdentifier
A handle to the object that was created by the merge.

in opaque directAncestorobjectIdentifier
A handle to the object which the new object is a direct descendent of.

in opaque mergedObjectList
A list of handles to the objects were merged into the “direct ancestor” component version.

Optional Arguments

in opaque libraryIdentifier
A handle for the library from which the object was checked out

in string version
The version of the module merged.

in string branchName
The name of the branch on which new object was created.

in string comment
The reason for the merge.

in namedvalues optionslist
A list of implementation specific options.

Merge(Request)

Merge multiple component versions, creating a new version

Description

The Merge message requests that the system merge the contents of a set of component versions creating a new version which is a descendent of the "direct ancestor" version. The new version is created in the checked out state. It is up to the version manager to handle conflict resolution. The component versions must be versions of the same component.

Required Arguments

out opaque objectIdentifier
A handle for the object that was created by the merge

in opaque directAncestorobjectIdentifier
A handle to the object which the new object will be a direct descendent of.

in opaque mergedObjectList
A list of handles to the objects which will be merged into the "direct ancestor" component version. Note that not all version managers support the merge of more than two component versions.

Optional Arguments

in opaque libraryIdentifier
A handle for the library in which the merged object should be created.

in string version
The version of the module to merge.

in string branchName
The name of the branch on which new object will be created. Note that this is only supported by version managers that support specifying a branch on checkout.

in string comment
The reason for the merge.

in namedvalues optionslist

A list of implementation specific options. The following options may be included:

out datastream contents

An argument that allows the contents of component to be accessed in an opaque manner.

Return Values

DIFF_COMPONENTS - The component versions are versions of different components.

ANOTHER_CHECK - The object is already checked out by another user

NO_CHECKOUT_PRIV - No privilege to checkout the component version

NO_LIBRARY - No version management library is set; set it with the Set_Environment request.

Fetch(Request)

Request that an object be fetched out of the system.

Description

The Fetch message requests that the system make available a copy of an object without locking the object.

Required Arguments

in opaque objectIdentifier
A handle for the object that will be fetched.

out datastream contents
An argument that allows the contents of component to be accessed in an opaque manner.

Optional Arguments

in opaque libraryIdentifier
A handle for the library from which the object should be fetched

in namedvalues optionslist
A list of implementation specific options.

Return Values

NO_FETCH_PRIV - No privilege to fetch the object

NO_LIBRARY - No version management library is set; set it with the Set_Environment request.

Create(Request)

Request that a particular kind of object be created.

Description

The Create message requests that a particular kind of object be created. The generic Created notification is sent if this request completes successfully.

Required Arguments

out opaque returnedObjectIdentifier
A handle for the object that was created.

in opaque typeIdentifier
A handle for the type of object that is to be created.

Optional Arguments

in string comment
A string indicating the purpose of the object

in opaque initialList
A list of initial attributes to set on the object. The contents of this list is dependent on the type of the object be created.

The following table describes the attributes for different types.

Table 14 Attributes for list

Object Type	Name	Type	Description
Component	objectName	string	The name of the component to be created.
Component	libraryIdentifier	opaque	A handle for the library in which the object should be created.
Library	location	opaque	The location in which to create the database.

optionsList opaque input

A list of implementation specific options. The following table describes the options for different types.

Table 15 Options for Different Types

Object Type	Name	Type	Description
Component	stream	opaque	An argument that allows the contents of component to be accessed in an opaque manner.

Return Values

OBJECT_EXISTS - Object already exists.

NO_CREATE_PRIV - No privilege to create the object.

Delete(Request)

Request that a particular object be deleted.

Description

The Delete message requests that an object be deleted.

Required Arguments

in opaque objectIdentifier opaque input
A handle for the object that is to be deleted.

Optional Arguments

in string comment
A string indicating the reason for the deletion

in string optionsList
A list of implementation specific options.

Return Values

NO_DELETE_PRIV - No privilege to delete the object.

Differences(Request)

Request that the differences between two version components be computed.

Description

The Differences message requests that the system compute the difference between two component versions. The component versions must be versions of the same component.

Required Arguments

in opaque objectIdentifier1
A handle for the first component version.

in opaque objectIdentifier2
A handle for the second component version.

out datastream contents
An argument that allows the differences between the components to be accessed in an opaque manner.

Optional Arguments

in opaque libraryIdentifier
A handle for the library in which the object are located.

in namedvalues optionslist
A list of implementation specific options.

Return Values

DIFF_COMPONENTS - The component versions are versions of different components.

NO_READ_PRIV - No privilege to read the object

NO_LIBRARY - No version management library is set; set it with the Set_Environment request.

CheckOut(Request)

Requests that an object be checked out.

Description

The CheckOut message requests that an object be made available for modification. The version manager locks the object on behalf of the requestor. If the component being checked out is a subcomponent of another component, the parent component(s) must also be checked out.

Required Arguments

in opaque objectIdentifier
A handle for the object that will be checked out.

out opaque returnedObjectIdentifier
A handle for the object that was checked out. The format of this handle is dependent on the version manager.

Optional Arguments

in opaque libraryIdentifier
A handle for the library from which the object should be checked out.

in string version
The version of the module to checkout.

in string branchName
The name of the branch on which the object will be checked out. Note that not all version managers handle specifying a branch on check out.

in string comment
A string indicating the purpose of the modification.

in namedvalues optionslist
A list of implementation specific options. The following options may be included:

out datastream contents
An argument that allows the contents of component to be accessed in an opaque manner.

in boolean concurrent
True if a concurrent checkout should be permitted. Note that this could mean another user has it checked out or the same user has it checked out.

in boolean checkoutParents
Also checkout parent components of this subcomponent (if they are not already checked_out).

Return Values

ANOTHER_CHECK - The object is already checked out by another user.

PARENT_COMPONENTS_CHECKEDIN - The specified component version cannot be checked out with also checking out its parent components.

NO_CHECKOUT_PRIV - No privilege to checkout the component version

NO_LIBRARY - No version management library is set; set it with the Set_Environment request.

CheckIn(Request)

Requests that an object be checked in.

Description

The CheckIn message requests that modifications to an object be saved in the version manager storage system and that the lock on the object is relinquished. If the component being checked contains a subcomponents, those subcomponents must be checked in before the parent component can be checked in.

Required Arguments

in opaque objectIdentifier
A handle for the object that will be checked in.

Optional Arguments

in opaque libraryIdentifier
A handle for the library to which the object should be checked in

in string version
The version of the module to checkin.

in string branchName
The name of the branch on which the object will be checked in. This may be required if a descendent the component version already exists. Note that not all version managers handle specifying a branch on check in.

in string comment
A string indicating how the object was modified.

in namedvalues optionslist
A list of implementation specific options. The following options may be included:

in datastream contents

An argument that allows the contents of component to be accessed in an opaque manner.

in boolean keep

If a stream was specified do not delete it when access to it is completed.

in boolean checkinSub

Also checkin the subcomponents of this component (if they are checked out).

Return Values

NOT_CHECKEDOUT - The specified version was not checked out

SUBCOMPONENTS_CHECKEDOUT - The specified component version cannot be checked in without also checking in its subcomponents.

VERSION_EXISTS - The specified object already exists; a branch must be specified

NO_CHECKIN_PRIV - No privilege to checkin the component version

NO_LIBRARY - No version management library is set; set it with the Set_Environment request.

`CancelCheckOut(Request)`

Request to cancel a previous checkout operation.

Description

The CancelCheckOut message requests that a lock for an object, resulting from a previous checkout operation be canceled. The checkout of a parent component cannot be canceled unless none of its subcomponents are checked out.

Required Arguments

in opaque objectIdentifier
A handle for the object whose checkout is to be canceled

Optional Arguments

in opaque libraryIdentifier
A handle for the library from which the object was checked out.

in string comment
A string indicating why the checkout is being canceled.

in namedvalues optionslist
A list of implementation specific options. The following options may be

in boolean force

Cancel the checkout, even if the checkout was done by a different user.

in boolean cancelSub

Also cancel the checkout of all the subcomponents of this component (if they are checked out).

Return Values

NOT_CHECKEDOUT - The specified version was not checked out

CHECKEDOUT_BY_OTHER - The specified version is checked out by another user

NO_CANCEL_CHECKOUT_PRIV - No privilege to cancel the check out

NO_LIBRARY - No version management library is set; set it with the Set_Environment request.

GetCheckOuts(Request)

Get a list of all objects checked out.

Description

The GetCheckOuts message requests a list off all the checkedout objects.

Required Arguments

out opaque checkoutList
A list of handles of reservation identifiers for all the objects that are checked out. To get information about each reservation the requestor must send the GetAttributes message. This will return information like comment, timestamp and username.

Optional Arguments

in opaque libraryIdentifier
A handle for the library whose checkouts are to be returned

in opaque usernameFilter
A filter that tells that version manager what usernames should be included in the results. By default checkouts of all users are returned. This filter must be of a form that the version manager recognizes.

in opaque timestampFilter
A filter that tells that version manager what time range should be included in the results. By default checkouts done at any time are returned. This filter must be of a form that the version manager recognizes.

in opaque filterList
A list of additional filters to use to restrict the result.

Return Values

NO_LIBRARY - No version management library is set; set it with the Set_Environment request.

NO_CHECKOUTS - No checkouts found making the specified criteria.

GetContents(Request)

Get the contents of an object.

Description

The GetContents message requests that the system return a list of all the objects that an object is composed of.

Required Arguments

in opaque objectIdentifier
A handle to the object to get the contents of.

out opaque returnedObjectList
A list of handles of objects that an object is composed of. For a library this will consist of a list of all the components in the library.

Optional Arguments

in namedvalues optionslist
A list of implementation specific options.

Return Values

NO_READ_PRIV - No privilege to read the contents of the object.

NO_CONTENTS - The object is not composed of other object.

GetAttributes(Request)

Request that a set of attributes be read on a specified object.

Description

The GetAttributes message requests that a set of attributes be returned for a specified object.

Required Arguments

out opaque objectIdentifier
A handle for the object whose attributes are to be read.

in opaque attributeList
A list of attributes name whose values are to be read.

out opaque attributeValueList
A list of attributes values corresponding to the attribute names.

Optional Arguments

in namedvalues optionslist
A list of implementation specific options.

Return Values

NO_READ_PRIV - No privilege to read the object.

`SetAttributes(Request)`

Request that a set of attributes be written to a specified object.

Description

The SetAttributes message requests that a set of attributes be written to a specified object.

Required Arguments

out opaque objectIdentifier
A handle for the object whose attributes are to be written.

in opaque attributeList
A list of attributes name whose values are to be written.

in opaque attributeValueList
A list of attributes values corresponding to the attribute names.

Optional Arguments

in namedvalues optionslist
A list of implementation specific options.

Return Values

NO_WRITE_PRIV - No privilege to write the object.