Unicenter® Service Desk

Web Services User Guide

r11.2

CA Product References

This document references the following CA products:

- Advantage™
- Allfusion®
- BrightStor®
- CleverPath™ Portal
- eTrust® Embedded Identity and Access Management (eTrust eIAM)
- Unicenter® Asset Management
- Unicenter® Asset Portfolio Management (APM)
- Unicenter® Management Portal
- Unicenter® Network and Systems Management (NSM)
- Unicenter® Remote Control
- Unicenter® Service Desk
- Unicenter® Knowledge Tools (KT)
- Unicenter® Software Delivery
- Unicenter® TNG for Windows

Contact Technical Support

For online technical assistance and a complete list of locations, primary service hours, and telephone numbers, contact Technical Support at http://ca.com/support.

Contents

Chapter 1: Web Services Basics

This document describes the web services interface for the Unicenter Service Desk and Knowledge Tools products. The Unicenter Service Desk Web Services are built upon the standards set by the World Wide Web Consortium (you can find this product on their web site), including SOAP and WSDL. These XML-based standards, combined with the well-known HTTP protocol, make Unicenter Service Desk functionality accessible to remote applications regardless of platform.

Web Services and Unicenter Service Desk Advantages

Web Services is a set of data exchange standards that enable communication between applications, even if they are on completely different platforms. This is analogous to browsing the Web on a personal computer— all remote websites are accessible regardless of whether they are hosted on Solaris, AIX, Windows, and so on. In the same manner, Web Services allow applications to communicate over HTTP to various servers regardless of platform. For example, a Microsoft Office application can communicate with a program on a UNIX server, and a Java Server Page can access a server hosted on a Windows server. This platform-neutral communication allows for powerful integrations.

The Web Services takes advantage of this technology, allowing most any application access to Unicenter Service Desk and Knowledge Tools. Web Services clients can create tickets, update assets, search the knowledge base, and much more.

Access Web Services

To access the Unicenter Service Desk Web Services (as with any web services), your application must be capable of handling standard web services technologies, including XML, SOAP, WSDL, and HTTP. The easiest method for implementing any web services is to use an environment or toolkit that supports these technologies. Many vendors and open-source initiatives offer SDKs and tools that enable a myriad of applications for accessing web services. Whether it is a C++ application, a Microsoft Office Add-In, a Java Server Page or a script, there is a tool available to add web services capabilities.

Some examples of environments and toolkits available include Microsoft's Visual Studio.NET, Microsoft's SOAP Toolkit, Java 2 Enterprise Edition technology, Sun's Java Web Services Development Pack, IBM Websphere, and Apache Axis. Many others are also available. We recommend that you browse the various options and select one that matches your requirements. Most toolkits provide documentation and samples for creating web service clients.

Implementations Offered

Two implementations of the Unicenter Service Desk Web Services were included in the prior release, one used Microsoft .NET and the other used Java 2 Enterprise Edition (J2EE) technology. However, only the J2EE implementation is offered in this release in order to provide consistent behaviors across platforms.

For backward compatibility, the post GA r6.0 J2EE Web Services is still offered in this release. Users of GA r6.0 .NET Web Services need to switch to J2EE Web Services. The WSDL of the post GA r6.0 Web Services is exactly the same as the original .NET implementation (meaning that code targeted for 6.0 should continue to function the same), but you will need to specify a different SOAP end point.

Important! After migrating from r6.0 Unicenter Web Services running on Unicenter Service Desk r6.0 to a r6.0-level Web Service on version r11, any object handles that you may have persisted should be considered invalid after migration.

Web Services Components

Web Services components for this release of Unicenter Service Desk vary from those required for previous releases of the product. The following topics summarize the components currently required.

Unicenter Service Desk Components

The installation files for this version of J2EE Web Service are initially installed by the Service Desk and Knowledge Tools installation and can be located in the following directory:

```
<NX_ROOT>/sdk/websvc/R11
```

The installation files for the backward compatible J2EE Web Services are located in the following directory:

```
<NX_ROOT>/sdk/websvc/60
```

where <NX_ROOT> is the root installation path for Unicenter Service Desk.

Obtain the Web Services' WSDL Document

The location you need to access to obtain the WSDL (Web Services Description Language) document depends on the release of Unicenter Service Desk you intend to use. The following outlines the two locations available.

r11

By default, the WSDL for the r11 Web Services resides at the following URL:

```
http://<servername>:<port>/axis/services/USD_R11_WebService?WSDL
```

Post GA 6.0

The WSDL for the post GA 6.0 Web Services resides at the following URL:

```
http://<servername>:<port>/axis/services/USD_ WebServiceSoap?WSDL
```

Note: Any client code written to the r6.0 Web Services will continue to function as long as it references the post GA r6.0 WSDL. The r11 interface is not compatible with the GA r6.0 interface.

Many servlet containers use a port other than 80. For example, Tomcat defaults to port 8080, which is established during installation.

Tips for Web Services Clients

To assist developers in getting started with web services client application development, there is a sample Java client application for the Web Services included in the *samples* directory of the Unicenter Service Desk installation.

Many of the Web Services methods require arrays as input parameters (for example, the method createIssue() permits an empty array for the 'propertyValues'). Sometimes these arrays are optional, but the service requires an empty array to be passed. As a tip for using Visual Studio .NET to access Web Services, specify an empty array with one of the following arrays:

C# language

```
String[] emptyArray = new string[0];
```

Visual Basic .NET:

```
Dim emptyArray As String() = {}
```

Similarly for Java

```
String emptyArray[] = new String[0];
```

Note: emptyArray can then be passed to array parameters that accept empty arrays.

Configuration

The Unicenter Service Desk Web Services may be configured with entries in special web configuration files. The names and descriptions of the configuration options (described in detail later in this chapter) are summarized as follows:

Option Name	Description
design_mode_stubs	Sets the Web Service to 'design mode' (Service Desk only).
require_secure_logon	Requires the login() and loginService() web methods to be called with a secure protocol, such as https.

Option Name	Description
require_secure_connection	Requires that every web method be called with a secure protocol.
disable_user_logon	Disables both login() and loginService() web methods, so only loginServiceManaged() can be used to log in.

Note: The configuration settings can be set in the deploy.wsdd file. For r11, this file is located in this subdirectory: <NX_ROOT>/sdk/websvc/R11. For post GA 6.0, it is located in this subdirectory: <NX_ROOT>/sdk/websvc/60. We recommend that you create a back up copy of either file before making any changes to it.

New configuration settings take effect when Unicenter Service Desk Web Services is redeployed. The following steps (taken directly from the Apache Axis documentation), redeploys the Unicenter Service Desk Web Services:

1. Open a command prompt and set the CLASSPATH environment variable to include the required Axis jar files, which are supplied in <NX_ROOT>/java/lib. For example, to set it on Windows, use the following:

   ```
   set AXISHOME=%NX_ROOT%\java\lib
   set classpath=
   %AXISHOME%\axis.jar;%AXISHOME%\jaxrpc.jar;%AXISHOME%\saaj.jar;%AXISHOME%\comm
   ons-logging.jar;%AXISHOME%\commons-
   discovery.jar;%AXISHOME%\wsdl4j.jar;%AXISHOME%\log4j-1.2.8.jar;%classpath%;
   ```

2. Change the directory to <NX_ROOT>/sdk/websvc/R11 (or <NX_ROOT>/sdk/websvc/60 for GA r6.0 Web Services), and run the following commands:

   ```
   java org.apache.axis.client.AdminClient undeploy.wsdd
   java org.apache.axis.client.AdminClient deploy.wsdd
   ```

3. Recycle Tomcat by recycling the Service Desk service. You may avoid shutting down the entire Service Desk system by recycling Tomcat by simply using the following commands:

   ```
   pdm_tomcat_nxd –c stop
   pdm_tomcat_nxd –c start
   ```

The Unicenter Service Desk Web Service is now redeployed. You can verify that the service actually deployed by viewing the Axis services listing page at the following default URL:

```
http://<servername>:<port>/axis/services
```

Note: The exact URL depends upon your installation settings.

Security

There are important security considerations in deploying web services. The default configuration when using HTTP is insecure, as it is for all information in web service calls sent between the client and the server in plain text over the network using the HTTP protocol. This includes not only application data, such as ticket descriptions and contact names, but also web service session identifiers (SID); and depending upon the web service application login methods used, it may include passwords. Administrators deploying web services are highly encouraged to review this section carefully and to take additional configuration steps at the application and network levels in order to secure their web service environment.

Important! The default web service configuration used with HTTP is insecure and vulnerable to security threats, which can include password discovery, session fixation, and data spying, among others.

There are three interrelated key security considerations in deploying Web Services:

- What (application level) access authentication schemes should this deployment support?
- What additional networking level security features does this deployment require?
- How will these requirements be enforced through web service configuration options?

The following describes each security feature:

Web Service Application Level Authentication Schemes:

To access Web Services, a web service client application must be authenticated with the web service application. Web Services provides two schemes of access authentication. The first is by username/password, and the other is by Public Key Infrastructure (PKI) technology. Both work closely with the Access Control and Management component in Web Services, using access policy. Access authentication and access management are the most important security features of Web Services. For more information about access authentication and access management, see the chapter “External Specifications”.

Authentication with username/password methods may be disabled using the following security configuration command:

```
disable_user_logon
```

Before enabling this option, the administrator needs to determine if each web service client for which an enterprise is requesting Web Services access, can actually provide support for the alternative authentication method, which is the PKI-based login method. The key advantage to the PKI technology is that Web Services client applications do not require *maintained* system user accounts, that is; the maintenance, storage, and transmission of their passwords.

Networking Level Security Configuration:

In both authentication schemes, username/password and Public Key Infrastructure (PKI), notice that the session identifier returned from the specific login method (as well as all subsequent information), are transmitted in plain text when using HTTP. Furthermore, if the username/password authentication scheme is used, the password is sent unprotected (in plain text) from the web service client application to the Web Services. During product development, the W3C did not have recommended standards for web services security. Subsequently, WS-Security is not used by these Web Services implementations to provide a security context. Instead, point-to-point transport layer security (SSL/TLS) and other network level security mechanisms (for example, IPSec), are recommended to protect the otherwise plain text transmission of the application-level authentication exchange(s), and subsequent session identification and data.

Important! SSL (or https) is recommended when deploying Web Services to protect the application-level authentication exchange(s) and subsequent transmissions of session identification and data.

Web Service Configuration:

To allow administrators to enforce communications protocol-level security at the level of the Web Services application, the following two security configuration commands are supported:

`require_secure_logon`

This security feature requires you to use SSL (or https) for calling the Login() and LoginService() methods. This feature also provides a handy method for protecting the username and password, while avoiding the overhead of SSL for the rest of the web services.

Important! If you use the require_secure_logon command, the Web Services application will not confirm that communications protocol-level security is enforced for methods other than Login() and LoginService(). Unless other precautions are taken, the other Web Services methods may be invoked insecurely, causing greater vulnerability to security threats.

`require_secure_connection`

This security feature requires you to use SSL to access any part of the web service. If https is required but not used, then a SOAP Fault with code UDS_SECURE_CHANNEL_REQUIRED is returned. For more information about how to configure SSL, see the J2EE Servlet Container documentation.

Error Handling

If an error occurs with a Web Services method, a SOAP Fault is returned. The SOAP Fault is the standard means of returning exception information for Web Services.

The Fault message contains standardized <Message> and <Code> elements, but the most informative is the <Detail> element. The <Detail> element contains <ErrorCode> and <ErrorMessage> elements. The <ErrorCode> element returns an enumerated error code specific to either the Unicenter Service Desk or Unicenter Knowledge Tools product. The <ErrorMessage> contains an English string describing the errors. The <ErrorMessage> elements are more suitable for aiding the developer and more appropriate messages should display to users.

For example, the following illustrates a SOAP Fault when a bad parameter is supplied to Unicenter Service Desk's getObjectValues() method:

```
<soap:Fault>

    <faultcode>soap:Client</faultcode>

    <faultstring>Error on fetch with attribute
    list:persistent_id,first_name,last_nameParamErrorHere<faultstring>

    <detail>

<ErrorCode>1001</ErrorCode>

<ErrorMessage> Error on fetch with attribute list:
persistent_id,first_name,last_nameParamErrorHere </ErrorMessage>
    </detail>
    </soap:Fault>
```

If you are using a client built with Microsoft .NET managed code, a failed Web Services method call raises a "SOAPException" exception. All errors cancel the operation invoked.

In some cases, errors may be written by the servlet container and therefore, display in the servlet container logs. In other cases, error information may be written to Unicenter Service Desk logs. These logs are located in the following subdirectories:

- In the /bopcfg/www/CATALINA_BASE/logs subdirectory of Unicenter Service Desk installation
- In the /log subdirectory of the Unicenter Service Desk installation and to all that have the prefix "stdlog" (see the following note concerning activating this logging)

Note: We recommend that you constantly monitor these logs, as the server may log its own errors without reporting them to the Unicenter Service Desk Web Services.

Web Services Installation

The Web Services is installed during Unicenter Service Desk installation for both the primary and secondary servers. For the primary server installation, the Web Services is automatically deployed during Unicenter Service Desk configuration. For the secondary server, this is an option you select during Service Desk configuration. All appropriate files are included with the primary server installation. If you want to install the Web Services on a machine other than the primary, install the primary package along with the web engine package, but configure it as a secondary. You do not need to activate the secondary object server or web engine. For detailed information, see the online help for Service Desk configuration.

Chapter 2: External Specifications

This chapter describes what you need to know for specifying user access authentication and the features available by access control and management.

User Access Authentication

Unicenter Service Desk Web Services provides two access authentication schemes. They are associated with the new access control and management feature, which uses an access policy (For more information about creating and defining an access policy, see Defining an Access Policy.)

User Name/Password

Verifies the User Name/Password, as described in previous releases of the product.

Public Key Infrastructure (PKI) Technology

Verifies that the person requesting the access has ownership of a certain private key.

Important! If you plan to use an application that accesses this version of Unicenter Web Services for Service Desk, we strongly recommend that you first define a Web Service Access Policy, complete with its code value, in Service Desk. A default access policy with a policy code of DEFAULT is available when Service Desk is installed and configured.

User Name/Password Authentication

If you plan to use the User Name/Password type of access authentication, the user application needs to invoke one of following two web services methods to gain access to Unicenter Service Desk Web Services.

Note: The login user that you specify in the username parameter (not the proxy contact specified in the policy) is responsible for activities initiated in a session. All function group security and data partition is enforced for this login user.

login (Username, Password)

This method is provided for backward compatibility, where access authentication is performed on the username and password supplied. A SID (session ID) is returned only if the access is authenticated. All subsequent web services calls need to include this SID. Default access policy is then applied to all subsequent web services accesses labeled with the SID.

Username and password are required fields that require plain text when you define them.

loginService (Username, Password, Policy)

This method is similar to the previous login function in that access authentication is performed on the username and password supplied. A SID is returned only if the access is authenticated. However, a specific access policy, as identified in the third parameter, is applied to control and manage all subsequent Web Services accesses. Empty content in the policy parameter automatically applies the default policy.

Username and password are required fields that require plain text when you define them. Policy is required, but can be empty, and you must use plain text. Use the policy code defined in a policy.

How a login is validated depends on the contact’s assigned *Access Type*. The Access Type object is hosted by Unicenter Service Desk and sets the validation type. To view the Access Type record in the Unicenter Service Desk Web Interface, access the Web Authentication tab. You can also use the getAccessTypeForContact() web method to retrieve any Access Type object information. For more information about Access Types, see the *Unicenter Administrator Guide.*

Public Key Infrastructure (PKI) Authentication

If you plan to use the PKI authentication, realize that the content of the login request is encrypted with a private key that can only be decrypted by its matching public key. The response of the login request is returned as plain text.

Generally, each application accessing Unicenter Service Desk Web Services is assigned with a policy. Unicenter Service Desk Web Services stores detailed information about a policy, along with the public key of a digital certificate. An application, as the policy holder, uses the private key of the digital certificate and the policy code (as policy identifier) to assemble a login request.

loginServiceManaged (Policy, Encrypted_Policy)

Unicenter Service Desk Web Services performs the user authentication by locating the policy through the plain text policy code, retrieving the policy holder’s public key associated with the policy, decrypting the encrypted policy code, matching the decrypted content with the policy code, and finally, opening a session with a back-end server. The plain text session ID (SID) is returned and can be used for subsequent method invocations. Only the policyholder holds the private key that matches the policy’s associated public key stored in Unicenter Service Desk.

All subsequent web services calls must include the returned session ID (SID). The Proxy contact specified in the policy is responsible for all web services activities initiated in this session. All function group security and data partition is enforced for the proxy contact.

Important! The Encrypted_Policy parameter should be in the BASE64 text format. The user application must perform proper conversion from the binary format.

Policy is a required field. When you define it, use plain text policy code as defined in a policy. Encrypted_Policy (the digital signature of the policy code encrypted with the policy holder’s private key) is required. When you define Encrypted_Policy, use the algorithm SHA1 with RSA to obtain the digital signature.

Configuration for the PKI Authentication Type

To configure for PKI authentication, you must first create an access policy. The process flow is as follows:

Create an Access Policy

The administrator performs this task by using the GUI (web interface only) provided by Unicenter Service Desk, and as part of the process, needs to assign a unique text code to each access policy. For additional information, see Defining an Access Policy.

Obtain a Digital Certificate with a Public/Private Key Pair and Associate it with the Access Policy

For PKI access authentication, a user application needs to obtain a digital certificate that contains both a public key and private key pair. An administrator can obtain the digital certificate through third-party Certificate Authority (CA) or security products that support digital certificates. Unicenter Service Desk also provides a server-side utility that can generate a digital certificate. It is located in <NX_ROOT>/bin directory as follows:

```
pdm_pki –p policy_code [–l certificate file] [–f]
```

If you obtain a digital certificate through a third party CA or security products, import it to where the Unicenter Service Desk server is located, and then associate it to an access policy. The administrator of the user application should obtain a digital certificate file that includes the content of an X509 V3 certificate in DER/ASN.1 format.

In addition, the certificate should contain only the public key of the public/private key pair. Using the –l option, the administrator should invoke the pdm_pki utility to load the certificate. The utility then loads the certificate, extracts the public key, converts the public key to BASE64 text format, and saves it with the access policy specified by the policy code.

When a digital certificate is generated by the pdm_pki utility, the administrator invokes the command in Service Desk without the –l option. The utility then generates a public and private key pair (keys are RSA1024 bit keys). The public key is converted to BASE64 text format where it is stored along with the access policy specified by the policy code. An X509 V3 certificate is also created to hold the public key along with other information (the default pass phase is set as the policy code). Finally, the X509 V3 certificate is packaged with the private key to a standard portable certificate format of PKCS12. It is then saved in a file with a file name of *policy_code*.p12, depending on the policy code supplied. This file can then be exported to clients.

Note: If an access policy has already been associated with a public key of a certificate, users need to specify the –f option when calling the pdm_pki command in order to overwrite the existing public key with a new public key.

Login to Web Services

The following describes the process flow for logging in to Unicenter Web Services configured with PKI authentication:

Process	**Description**
Load the Digital Certificate and Extract the Private Key	The digital certificate must be stored in secure storage on the user side, where it can be retrieved and used for logging in to Unicenter Web Services. Example of secure storages include the following: ■ Windows Certificate Store ■ Java Certificate Store (managed by java_keytool utility) ■ Certificate store (created by other security products, such as one of the eTrust products). A user application should be able to load the digital certificate and extract the private key using appropriate APIs, depending on user environments.
Create a Digital Signature of the Plain Text Policy Code with the Private Key	Once the private key is extracted from the digital certificate, it can be used to generate a digital signature of policy code. Creating a digital signature encrypts a digest of a text with a private key. The digest algorithm must be standard SHA1, and the encryption algorithm should be RSA. Also, the binary digital signature should be converted to BASE64 text format before it can be used for logging in to Unicenter Web Services. Depending on user environments, appropriate API calls should be used to archive this information.

Process	Description
Invoke the Web Service Call	A user application should invoke the Web Services method loginServiceManaged(), along with the plain text policy code and the BASE64 text formatted digital signature of the policy code.
Obtain the Returned SID	If the access request is authenticated, a plain text SID is automatically returned.

Once a SID is generated, it establishes a successful binding between a Web Service session and an access policy. The user application can invoke other web services methods with this SID, and all of its access to Unicenter Web Services becomes controlled and managed by this access policy.

Session and Authorization

A successful validation returns a SID that is associated with the validated username, whether it is the user name supplied for login or the proxy contact specified in a policy. Because of this process, each Service Desk user is assigned security rights that you may want enforced in your web service application.

For example, a specific user may have a Data Partition restricting which Requests the user can view. When using a SID for the user to get Request information, the Service Desk system ensures the data partition is enforced.

Function Group security is also applied. For example, a user may not have access to the Call Manager function group. Invoking any web services methods, such as viewing or creating Requests, is denied because access is denied to the Call Manager function group.

When your application is finished doing work for a user, call the Logout() method to invalidate the SID.

Each SID expires after a period of inactivity. That is, a SID expires if the interval between method calls is greater than a certain timeout value. The timeout interval is set in Options Manager and is specified by the following Service Desk Option:

```
‘webservice_session_timeout’
```

If this value is set to zero (0), a SID never times out. If this option is missing or not set, the default is one hour. If a Web Service method is called with an expired SID, a Fault is returned with an error code of UDS_SESSION_TIMEOUT the first time it is referenced, and UDS_BAD_SESSION each time thereafter.

To keep a SID active, call any web service method before the time out is reached. To keep the SID active without working the server, call serverStatus().

Access Control and Management

To minimize the potential problem of web services ticket flooding and to maintain the stability of the Service Desk server, this version of Unicenter Service Desk Web Services uses an Access Control and Management system. It works primarily to handle the excessive service activities initiated by trusted user applications that can result from programming errors or exceptions. It also works as a barrier for controlling access to Unicenter Service Desk Web Services from malicious attackers. An administrator of a web service application is able to create and define an access policy in Unicenter Service Desk that controls access to Unicenter Service Desk Web Services from a web service application.

Note: A default access policy is provided out-of-the-box and has a code of DEFAULT. The default access policy contains no access restrictions and is only applied to sessions authenticated through username and password.

Define an Access Policy

To create any web services access policy, an administrator has to define an access policy. Perform the following steps to define the access policy:

1. Launch Unicenter Service Desk and select the Administration tab to view the Administration menu.
2. Select Web Services Policy, and then Policies, to access the Create New Web Services Acess Policy window
3. Each access policy contains a unique symbol and code. Enter field information based on the following table descriptions:

Field	**Description**
Symbol	(Required) Identifies a symbolic name of the access policy.
Code	(Required) Indicates the unique text that identifies this access policy.
Status	Identifies the status of an access policy. An inactive policy is not used.
Proxy Contact	Identifies the contact to use for all web services operations and Service Desk security.

Field	Description
Default	Identifies the default policy. Set this policy as the default policy. Only one active default policy is allowed to exist. Creating a new default policy automatically sets the current default policy to a non-default status.
Has Key	(Read-Only) indicates whether a public key has been associated with this policy. This field is updated when a public key is associated with a policy through the pdm_pki utility.
Allow Impersonate	Identifies the allow impersonate privilege. if this field is set, the policyholder can invoke the impersonate() web services method and create a new web services session in the name of the user to be impersonated. Additional access authentication is not performed when creating the new session. However, only when the access_level of the new user's access type is less than or equal to the grant_level of the proxy user's access type, can this method be successfully called.
Description	Indicates the detailed description of this access policy
Ticket Creation	Indicates the number of ticket (call request, change order, and issue) insertion operations allowed per hour.
Object Creation	Indicates the number of Service Desk object (other than ticket object) insertion operations allowed per hour.
Object Updates	Indicate the number of Service Desk object update operations allowed per hour.
Attachments	Indicates the number of attachment-related operations allowed per hour.

Field	Description
Data Queries	Indicates the number of data query operations allowed per hour.
Knowledge	Indicates the number of knowledge-related operations allowed per hour.

Note: The default value of -1 in any operation counter indicates that no restrictions apply to the corresponding operation. A value of 0 (zero) indicates that the corresponding operation is not allowed.

4. Click Save to save the access policy.

Web Services Methods by Categories

Each Unicenter Service Desk Web Services method belongs to a specific category. The following lists each category and their corresponding methods:

Ticket Creation

- createTicket()
- createQuickTicket()
- createRequest()
- createChangeOrder()
- createIssue()

Object Creation

- logComment()
- createAsset()
- addAssetLog()
- createAssetParentChildRelationship()
- createObject()
- createWorkFlowTask()
- createActivityLog()
- notifyContacts()
- addBookmark()
- addComment()
- createFolder()

Object Updates

- addMemberToGroup()
- removeMemberFromGroup()
- closeTicket()
- createLrelRelationships()
- removeLrelRelationships()
- deleteWorkFlowTask()
- updateObject()
- transfer()
- escalate()
- attachChangeToRequest()
- detachChangeFromRequest()

- changeStatus()
- clearNotification()
- updateLrel()
- deleteBookmark()
- updateRating()

Attachments

- createAttmnt()
- createAttachment()
- attachURLLink()
- deleteAttmnt()
- deleteComment()
- removeAttachment()

Data Queries

- impersonate()
- serverStatus()
- getBopsid()
- getConfigurationMode()
- getHandleForUserid()
- getAccessTypeForContact()
- getPermissionsGroup
- getObjectTypeInformation()
- getRelatedList()
- getRelatedListValues()
- getGroupMemberListValues()
- getPendingChangeTasksForContact()
- getPendingIssueTasksForContact()
- getWorkFlowTemplates()
- getWorkflowTemplateList()
- getTasksListValues()
- getNotificationsForContact()
- getPolicyInfo()
- getAssetExtensionInformation()
- getLrelValues()

- getObjectValues()
- doSelect()
- doQuery()
- getPropertyInfoForCategory()
- getValidTaskTransitions()
- getListValues()
- getListInfo()
- findContact()
- getAttmntInfo()
- getAttmntList()
- getBookmarks()
- getCategory()
- getComments()
- getContact()
- getDecisionTrees()
- getDocument()
- getDocumentTypes()
- getFolderInfo()
- getFolderList()
- getLrelLength()
- getPriorities()
- getRepositoryInfo()
- getStatuses()
- getTemplateList()

Knowledge

- createDocument()
- deleteDocument()
- doSelectKD()
- faq()
- attmntFolderLinkCount()
- getAttmntListPerKD()
- isAttmntLinkedKD()
- getDocumentByIDs()

- getKDListPerAttmnt()
- getQuestionsAsked()
- modifyDocument()
- rateDocument()
- search()

When an access policy is updated by Unicenter Service Desk, Web Services dynamically updates the corresponding policy information. Active Web Services sessions controlled under this policy remain controlled with configurations in the policy. New Web Services sessions for this policy to manage and control, take the latest configurations in effect.

Note: For more information about each method, see the chapter "Web Services Methods".

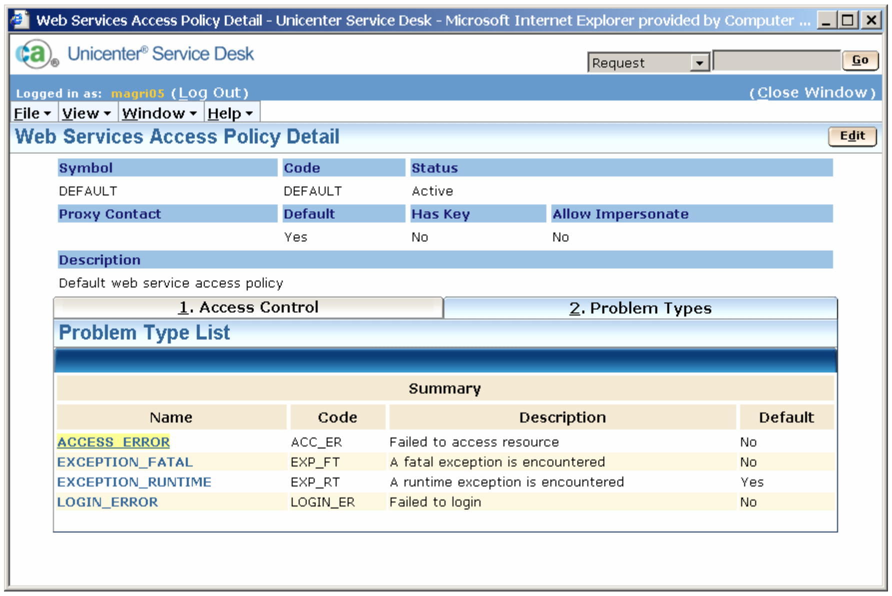
Define a Problem Type

Problem Types are assigned when creating tickets and an access policy defines one set of these problem types. A Unicenter Service Desk Web Services user application may use low-level web methods to create a ticket (request, change order or issue), specifying one of these types to categorize the problem addressed in the ticket. Problem Types can be used only with the high-level createTicket() method. For more information about high-level web methods, see Simplified Web Services Access.

Note: Low-level methods, such as createRequest(), do not use problem types.

Web Services Problem Types

Unicenter Service Desk Web Services also provides a defined set of out-of-the-box problem types, which are created for *every* policy. These default types, designated as *internal* problem types, can be deactivated, but never deleted. The following Web Services Access Policy Detail window shows the default problem types provided when a new policy has been created:


Web Services Access Policy Detail - Unicenter Service Desk - Microsoft Internet Explorer provided by Computer ...
Unicenter® Service Desk
Request
Go
Logged in as: magri05 (Log Out)
(Close Window)
File
View
Window
Help
Web Services Access Policy Detail
Edit
1. Access Control
2. Problem Types
Problem Type List

Symbol	Code	Status	
DEFAULT	DEFAULT	Active	
Proxy Contact	**Default**	**Has Key**	**Allow Impersonate**
	Yes	No	No

Description
Default web service access policy

Name	Code	Description	Default
ACCESS_ERROR	ACC_ER	Failed to access resource	No
EXCEPTION_FATAL	EXP_FT	A fatal exception is encountered	No
EXCEPTION_RUNTIME	EXP_RT	A runtime exception is encountered	Yes
LOGIN_ERROR	LOGIN_ER	Failed to login	No

The following describes each of the internal problem types:

ACCESS_ERROR

Indicates that the system failed to connect to or find a resource, such as a file, website, and so on.

EXCEPTION_FATAL

Indicates that the application is shutting down unexpectedly.

EXCEPTION_RUNTIME

Indicates that the application code encountered an exception.

LOGIN_ERROR

Indicates that the operator failed to gain access to the application.

Additional Problem Types

The administrator to an access policy can add additional problem types that are described in the following table:

Problem Type	Description
Ticket Template	Identifies a template of a request (Incident or Problem for ITIL), issue, or change order that you use to create a ticket when this problem type is reported. **Note:** The owning policy's contact is used as the end user. The Ticket Type and Ticket Template Name define the ticket template.
Default	Indicates if this problem type is the default for the policy. Only one default is allowed per policy. Note that a new default problem type overwrites the existing default problem type associated with the policy.
Active	Represents an active problem type. **Note:** An inactive type does not create tickets.
Internal	Identifies the field as read-only, which indicates whether this problem type is an internal out-of-the-box default problem type.
Symbol	Indicates the symbolic name of the problem type.
Code	Identifies the unique text identifier of the problem type.
Description	Describes the detailed description of the problem type.

Problem Type	Description
Duplicate Handling	Defines the action to take when the system detects that an identical ticket already exists. (For more information about Handling duplicate tickets, see Handling Duplicate Tickets).
Return Data	Identifies the user-defined return message you can specify for the web method “createTicket()” to return to client applications. Return data might be used for indicating an action the application should take (Application Data Return), or a message (User Data Return) to display to the end user.

Duplicate Ticket Handling

The Web Service Access Policy can detect and handle duplicate tickets, which is helpful for preventing ticket flooding. A ticket created with the potential of being a duplicate applies if all of the following conditions are true:

- At least one ticket of the same type (cr, iss, or chg) already exists and is ACTIVE.
- The existing ticket was created by the web service.
- The existing ticket was created with the same Policy and Problem Type as the ticket being created.
- The “create date” of the existing ticket is within a specified threshold (for example, it was opened less than 2 days ago).

  Note: The create date field is configured using the Maximum time interval for searching duplicates.
- The duplicate ID matches the one provided by users when invoking the createTicket() method.

Users can also assist in preventing duplicates by classifying tickets as being unique or different, based on criteria known to the user. To do this, add an optional string parameter to the createTicket Web Services call. If duplicate handling is on, the string parameter is inspected after other duplicate handling criteria match to determine whether this is a unique or duplicate call to this method.

Resolve a Duplicate Ticket

If the create ticket action results in a duplicate, the existing Problem Type may be configured to do one of the following:

Reconfigured Problem Type	Results
Create the New Ticket and Ignore Duplicates	A new ticket handle and number will be returned (default).
Do Not Create a New Ticket; Add an Activity Log to the Existing Duplicate Instead	The ticket handle and number of the existing ticket will be returned.
Do Not Create a New ticket; Add an Entry to the Service Desk Standard Log Instead	A ticket handle and number of the existing ticket will be returned.
Create a New Ticket and Attach it as a Child to the Duplicate	A new ticket handle and number will be returned.

Simplified Web Services Access

Unicenter Service Desk Web Services provides an abbreviated set of high-level web services methods that are simplified versions of existing web services methods. The majority of users applications do not have to completely rely on a large set of web services methods before requesting service desk services through Unicenter Service Desk Web Services. Working closely with user-defined access policies and using default parameters defined in the policies, this set of high-level web services methods can function with little knowledge of the Service Desk object schema. Also, the high-level methods cover a common set of Service Desk functionalities that most ServiceAware applications need.

The following describes the use of these high-level web services methods:

createTicket (SID, Description, Problem_Type, Userid, Asset, DuplicationID)

You must specify a problem type for the reported problem if you use this method. The problem type should contain the ticket template appropriate for the ticket you want to create. It should define the action to take in the case of a duplicate ticket, specify the data outputs, and finally, it must be associated to the access policy that is defined for the user application.

When this method is invoked, Unicenter Service Desk Web Services locates the current access policy and the problem type required for the ticket creation. The following shows the sequence that Unicenter Service Desk Web Services uses for locating the proper problem type:

- If a specific problem type code is provided as input and it matches a problem type that is associated to the policy, this problem type is used, regardless of whether it is internal.
- If a problem type is not specified or the previous step fails to locate a problem type, the default problem type is used if there is one defined for the policy.
- If a default problem type is not defined for the policy or the previous step fails, the default problem type defined for internal problem types is used.

Once a problem type is defined, Unicenter Service Desk Web Services uses it to create a ticket. The proxy user defined in the access policy is used for the ticket creation if the userid is empty, and asset information is added to the ticket (if the input is not empty). Once the ticket is created successfully, Unicenter Service Desk Web Services returns both user data and application data, as specified by the problem type.

closeTicket (SID, Description, TicketHandle)

Users can call this function to close an open ticket. It simply sets the status of an open ticket to 'close' and adds the input description to the activity log.

logComment (SID, TicketHandle, Comment, Internal_Flag)

Adds an entry with the input comment to the activity log for the open ticket.

getPolicyInfo (SID)

Lets users obtain the policy information that controls the current web services session. You can use this information as an indicator of server capacity for this user application. Users may want to adjust their web services calls to fit into the capacity.

By having this set of simplified Web Services APIs, a majority of users are spared the tremendous effort of understanding the complete set of web services API and Service Desk schema. Using them simplifies and accelerates the process of creating ServiceAware enabled applications for these users.

Chapter 3: Unicenter Service Desk Web Services

This section explains additional Unicenter Service Desk Web Services behaviors.

Objects

The Unicenter Service Desk system treats each entity, such as a contact or an issue, as an *object*. These high-level objects are defined in majic (.maj) and mod (.mod) files on the Service Desk server in the following directory:

```
/bopcfg/majic
```

Customized objects are defined in the following directory:

```
/site/mods/majic
```

Objects are essentially high-level wrappers around a database table.

Important! The Unicenter Service Desk Modification Guide is essential to working with Web Services, especially the sections describing the objects and database schema.

An object's type (sometimes referred to as *factory*) defines the object. For example, request objects belong to the 'cr' type. Each object's type is defined by the "OBJECT" declaration in a majic file. All objects shipped with Unicenter Service Desk are enumerated in the *Unicenter Service Desk Modification Guide*.

An object has *attributes*, which are essentially columns in a database table (do not confuse these with XML attributes). Web Services offers many methods to retrieve values for attributes. Many methods require you to name attributes for setting or retrieving values. You must use the attribute name assigned in the majic or mod file that defines the object, which can be different than the actual database name. For a list of all the attributes for each object, see the *Unicenter Service Desk Modification Guide.* It is possible for client sites to add additional attributes as a customization.

Web Services uniquely identifies an object by its *handle,* which is a string value of the form *objectType:ID*, where *objectType* is the object's type (factory) name, and ID is a unique value. The ID value matches that of the 'id' attribute found in every Unicenter Service Desk object. Because the 'id' attribute is almost always indexed in the DBMS, using the ID portion of the object handle is especially valuable for forming efficient queries. Each object, regardless of its type, stores this value in an object attribute named "persistent_id".

Note: In prior releases, the ID portion of the handle was always a string of integers. In r11, the ID portion may also be the string representation of a UUID, typically 32 characters.

The following table lists the object and factory names of the entities that use UUIDs:

Object Name	Factory Name
Contact	cnt
Asset	nr
Organization	org
Location	loc
Company/Vendor	ca_cmpny
Model	mfrmod

Handles are *persistent;* a handle representing a particular object is always unique for its lifetime, even across database migrations. Clients may want to take advantage of this persistence when working with fairly static objects, for example, Status or Contact Types.

Object handles are the key to successfully using Service Desk Web Services. Many methods, especially those that update data, require handles. Most methods that return object data also include the object's handle.

Lock Errors

Unicenter Service Desk objects are locked during updates. Methods that update objects (such as, updateObject() or transfer()) may return the following lock error code:

```
UDS_LOCK_ERR
```

This code indicates that another user is updating the record. Often the locking user's handle is returned in the ErrorMessage element.

Access the Unicenter Service Desk Web Services

The Unicenter Service Desk Web Services uses the Apache implementation of standards set forth by the W3C. Ideally, a client on any type of platform should be able to access the services, but vendor implementations vary. Many programming environments provide a tool to generate proxy classes from a WSDL service description.

Time Outs

A method may take a long time to process if the Unicenter Service Desk server is heavily loaded. In rare cases, a method may never return because a separate process failed to reply or some other error occurred. To guard against excessive blocking, every Web Services method times out after a number of seconds. Web Services method time-out is a Unicenter Service Desk server time-out, *not* a Web server time-out, network time-out, and so on.

If a method times out, it returns the following error code:

```
UDS_TIMEOUT_ERR
```

The operation is not aborted! The server may have received the request and will process it successfully, although slowly.

Note: The Web Services will delay a few seconds the first time it is accessed after the J2EE application server is recycled. This happens because the application is initializing, loading DLLs, libraries, and so on, and occurs only with the first Web Services method call. All subsequent calls return much faster.

System Updates and Caching

The Unicenter Service Desk Web Services caches information for object types. Type information is not cached until the type is referenced for the first time, and will cause a small delay.

To avoid any server or caching delays, you may want to run a primer client to activate the Web Services and cache the most popular object type information. The easiest way to cache the object type information is to perform repeated calls to GetObjectTypeInformation(). The object types to consider for this technique could be one of the following:

Object Type	Definition
cr	Request
chg	Change Order
iss	Issue
cnt	Contact
nr	Asset
wf	Workflow (Change Orders)
iss_wf	Workflow (Issues)
prp	Property (for Change and Issue)
prptpl	Property Template (for Change and Issue)
cr_prp	Request Property
cr_prptpl	Request Property Template

Add any additional object types your client code references.

Categories and Properties

The request, change order and issue objects all have a category field, which is used to classify the nature of the ticket. A category may have zero or more property objects, which are instantiated and attached to the ticket when the category is assigned. Zero or more of these may be marked *required*, which means a value must be supplied before the ticket can be saved (and applies to both insert and update operations).

This enforcement works fine in a GUI environment, but is more difficult to deal with programmatically. To make things easier, the Unicenter Service Desk Web Services automatically supplies default values for any ticket created with the Web Services. The default value (currently, “-“) is obtained from Unicenter Service Desk’s localized message catalog.

If you need to set property values at creation time, there are three ticket creation methods: createChangeOrder, createIssue, and createRequest. Each has a parameter with which you can pass in values for any properties. To discover which properties will be attached, you must find out the properties associated with the category you intend to assign to the ticket. The easiest method to use is getPropertyInfoForCategory(). For more information about the getPropertyInfoForCategory(), see createRequest().

To set property values after an update operation with updateObject(), you must query the property list after the update. getRelatedList() can help with this task.

Design-time Feature

The Unicenter Service Desk Web Services includes a method stub configuration feature for developers. When activated, the Web Services ignores the Unicenter Service Desk server and returns simulated data for method calls so Web Services calls can be made without running a Unicenter Service Desk server.

To activate this feature in the Java version, edit deploy.wsdd to uncomment the sections for “design_mode_stubs”. You must reverse the deployment and redeploy the server, and recycle the application server.

Note: The Design-time feature applies to Unicenter Service Desk Web Services methods only.

XML Object Returns

Many of the Web Services methods return an XML representation of Unicenter Service Desk objects. The Web Services uses a standard XML structure beginning with the following root element:

```
<UDSObject>
```

The format of the XML representation is described in the following table:

XML Element	**Type**	**Description**
<UDSObject>	N/A	Identifies the root node.
<Handle>	String	Identifies the object's handle.
<Attributes>	Sequence	Identifies the attribute values. This holds zero or more elements for the object's attribute values.
<*attrName0* DataType = "*typeEnum*">	String	Identifies the *AttrName0,* which is an object attribute name as defined in the Service Desk majic (.maj) or mod (.mod) file. This name may use dot-notation depending on the web method used. The element's value is the attribute's value. An empty element indicates a null/empty value for this object's attribute. The DataType attribute is an integer indicating the attribute's data type in the Service Desk environment. The value is one of the data type enumerations described in the Data Types section in this document.

For example, a call to getObjectValues() can return information illustrated by the following:

```
<UDSObject>
        <Handle>cnt:555A043EDDB36D4F97524F2496B35E75</Handle>
        <Attributes>
                <Attribute DataType="2003">
                        <AttrName>first_name</AttrName>
                        <AttrValue>first name</AttrValue>
                        <DisplayValue>Yaakov</DisplayValue>
                </Attribute>
                <Attribute DataType="2005">
                        <AttrName>organization</AttrName>
                                <AttrValue>342</AttrValue>
                <DisplayValue>Accounting Crew</DisplayValue>
        </Attribute>
</Attributes>
        <Lists>         <List name="mylist1">
                        <UDSObject>...</UDSObject>
                        <UDSObject>...</UDSObject>
                </List>
        </Lists>
</UDSObject>
```

Some methods, such as doSelect(), return a sequence of <UDSObject> elements contained inside a <UDSObjectList> element.

The <Lists> section holds zero or more <List> nodes. A <List> node holds zero or more <UDSObject> nodes. <List> elements are generally returned only when a specific request for list values is made.

For the generic data retrieval methods, such as getObject(), a list is requested by naming it in the return value array. The list requested must be a BREL/QREL/LREL defined for the object. The list is requested in the parameter array of the form "*listname.attribute",* where listname is the name of a QREL/BREL/LREL (for example, for the Request object, actlog.description, actlog.time_spent, and so on). Requests for multiple values from the same list source are consolidated into a single <List> return (such as, the previous example should only return one <List> element for the actlog list). The contents are <UDSObjects> with two attributes, time_spent and description.

If a request is made just for a list with no attribute name, such as actlog, then the entire <UDSObject> is returned in the <List> section.

Specialized methods, like getDocument(), can of course be different. When a request is made for an attribute, the database value is returned. For SREL attributes, this may not be so useful. Requesting the assignee attribute of a Request returns an integer since the Contact REL_ATTR (foreign key) is its ID. For Unicenter Service Desk, r11, the return data for attributes includes elements for the DBMS and common name value of SREL references.

Note on Using the ITIL Methodology Installation

The Web Services fully supports the ITIL methodology if the Unicenter Service Desk installation is configured to support the ITIL. The Web Services API uses terminology inherent to the out-of- the-box product, but you only need to follow the simple guidelines in the following sections to take advantage of Unicenter Service Desk's ITIL features.

For more information about ITIL methodology, see the *Unicenter Service Desk ITIL User Guide*.

Create an Incident or Problem

When Unicenter Service Desk is configured to use the ITIL methodology, Incidents and Problems can easily be created using the Unicenter Service Desk Web Services. Both Incidents and Problems are held in the cr (Call_Req) object. One of these attributes, named *type,* distinguishes the record as a request, incident or problem. To create a request, incident or problem, call createRequest and specify the appropriate value for the type attribute.

The type attribute is a pointer (SREL) to the crt (Call_Req_Type) object, so you must pass a handle as the value. For more information about possible crt handle values, see Out-of-the-Box Handles.

For example, the following pseudo-code illustrates how to create a new incident by passing the correct crt object handle to the createRequest method. In the following example, setting the *type* attribute in the name-value pair that is passed as a parameter to createRequest, creates the new incident:

```
attrVals = {“summary”, “A new incident”, “description”, “new incident”, “type”,
“crt:182”}

USPSD.createRequest(SID, creatorHandle, attrVals, template, new String[0], new
String[0])
```

Note: The *type* attribute is only used when Unicenter Service Desk is configured for ITIL, so it can be ignored if ITIL is not being used.

Query for Incidents or Problems

To retrieve incidents or problems, include the *type* attribute of the cr object in the where clause. The following example illustrates a where clause for retrieving all active Incidents. This where clause could be used with methods that perform queries for ‘cr’ objects, such as doSelect and doQuery:

```
type.id = 182 AND active = 1
```

The ‘182’ is the ID portion of the handle representing Incident types (for more information, see the crt (Call_Req_Type) objects table illustrated in Out-of-the-Box Handles. For more information about forming proper queries, see Where Clauses.

Attach an Incident to a Problem

The Web Services can create associations between ITIL tickets, such as associating one or more Incidents to a Problem. The parent attribute on a ticket is used to create parent-child relationships between the cr objects (which are used for request, incident and problem objects). Setting the parent attribute of a ticket to point to another ticket creates the relationship.

For example, a newly created incident relates to an existing problem. To associate the incident to the problem, use UpdateObject to set the incident’s parent attribute to point to the problem. The following pseudo-code example illustrates this by setting the parent attribute to the handle of an existing problem ticket:

```
attributeValues = {“parent”, “cr:12346”}
USPSD.UpdateObject(SID, incidentHandle, attributeValues, new String [0])
```

Attach a Problem to a Change Order

Incidents and problems can be linked to change orders with the attachChangeToRequest method. The following pseudo-code example uses this method to simultaneously create a new change order and attach it to a problem. In the example, “cr:12347” is the object handle of the problem—it passes a blank handle for the fourth parameter, which causes the method to create a new change:

```
UPSPSD.attachChangeToRequest(SID, creatorHandle, “cr:12347”, “”, new String[0],
“activity description”)
```

Configuration Items

The ITIL methodology uses the term Configuration Item to refer to software, hardware, and other IT resources. The out-of-the-box Unicenter Service Desk Web Services uses the term Asset. These terms both refer to the “nr” object stored in the ca owned resource database table. All methods described in this guide that use asset objects, specifically those described in the Web Services Asset Management Methods and Web Services Business Methods sections, work in the same manner for Configuration Items. It is merely a difference in terminology.

Use the Web Services

The information in this section provides you with the fundamentals for using the Unicenter Service Desk Web Services. Example code using the Web Services exists in the following Service Desk installation directory:

```
<NX_ROOT>/samples/sdk/websvc/java
```

The sample code is written in Java using Apache Axis for SOAP messaging.

Logins

Before any Web Services method can be used, a SID (session ID) must be obtained from one of these methods: login(), loginService(), and loginServiceManaged(). The first two methods require a username and password that are validated exactly the same as the Unicenter Service Desk Web client; the contact’s Access Type specifies the validation method. The third method requires a public/private key pair, where login request encrypted with the private key can only be decrypted through the public key, and vice versa. For more information about access authentication, see “External Specifications”.

Perform Common Tasks

The Web Services is a flexible and powerful API into the Unicenter Service Desk, but it requires some knowledge of the system's object structure. The first step is to familiarize yourself with the appendix "Objects and Attributes" in the *Unicenter Service Desk Modification Guide*. It lists the attributes of each object in the system, which is essential because many of the Web Services methods require attribute names.

The next step is to review the Web Services methods, especially generic ones. For example, if your application needs to display all the activity logs for a request, first identify how the activity logs relate to the request. The *Unicenter Service Desk Modification Guide* shows that the request object has two lists of Activity Logs: The act_log (which shows only non-internal logs), and act_log_all (which lists all activity logs).

Finally, identify which Web Services methods are needed. To get lists attached to an object, use getRelatedList() or getRelatedListValues().

Out-of-the-Box Handles

Some data shipped out-of-the box is used frequently. Instead of looking up handles for these objects, some of the commonly used ones are listed in the following tables.

Note: While the handles do not change, the legible symbols may be edited.

Contact Type (Object name: ctp)

Handle	Note
ctp:2307	The "Analyst" type
ctp:2310	The "Customer" type
ctp:2305	The "Employee" type
ctp:2308	The "Group" type

Impact (Object name: imp)

Handle	Note
imp:1605	Impact 'None'
imp:1600	Low impact '5'
imp:1601	Medium-low impact '4'
imp:1602	Medium impact '3'

Handle	Note
imp 1603	Medium-high impact ‘2’
imp:1604	High impact ‘1’

Priority (object name: pri)

Handle	Note
pri:505	Unassigned priority ‘None’
pri:500	Low priority ‘5’
pri:501	Medium-low priority ‘4’
pri:502	Medium priority ‘3’
pri:503	Medium-high priority ‘2’
pri:504	High priority ‘1’

Severity (object name: sev)

Handle	Note
sev:800	Low severity ‘1’
sev:801	Medium-low severity ‘2’
sev:802	Medium severity ‘3’
sev:803	Medium-high severity ‘4’`
sev:804	High severity ‘5’

Call Request Type (object name: crt)

Handle	Note
crt:180	Request
crt:181	Problem—available only with the ITIL installation
crt:182	Incident—available only with the ITIL installation

Query for Requests (or Issues or Change Orders) Assigned to a Contact

One of the most common operations is retrieving the active requests assigned to an analyst (assignee). You can use one of several methods, such as doQuery() (to get a list reference), or doSelect() (to get the values immediately). Assuming the assignee's handle is already known, the where clause to use is as follows:

```
assignee.id = U'<assigneeID>' AND active = 1
```

where

`<assigneeID>` is the id portion of a contact handle, value, such as "555A043EDDB36D4F97524F2496B35E75".

This where clause works for requests, change orders and issues because they all have the 'assignee' and 'active' attributes, and they mean the same thing for all three object types. The 'active = 1' portion of the where clause restricts the search to active requests. For more information on the active flags, see the next section.

The Active Flag

Most Unicenter Service Desk objects have a field called 'active' or 'delete_flag'. This is actually an SREL pointer to the Active_Boolean_Table object or Boolean_Table object. Consider adding these fields to your queries to filter objects marked as Inactive by the system administrator. For querying purposes, search for 'delete_flag = 0' to find active records and 'delete_flag = 1' for inactive records. For example, the following pseudo-code demonstrates using doSelect() to retrieve values for all active Request Status objects:

```
doSelect(SID, "crs", "delete_flag = 0", -1, new String[0]);
```

To set an object to active or inactive, you need to pass the handle of the Boolean object representing either true or false. These handles do not change, so you can safely hard-code them. These are listed as follows:

Active_Boolean_Table	**Boolean_Table**
actbool:4551 = 'Active'	bool:200 = 'False'
actbool:4552 = 'Inactive'	bool:201 = 'True'

Retrieve Related List Length

When requesting attribute values from an object, such as with getObjectValues(), you can get the length of a related list by requesting the following attribute:

```
“<listName>.length”
```

For example, to get the number of Activity Logs for a certain request, pass the following to getObjectValues():

```
“act_log_all.length”
```

Note: This is the only way you can use list names in these types of methods.

Error Codes

The following table lists the possible values for the <ErrorCode> value in a SOAP Fault returned from a Web Services call:

Error Name	Value	Description
UDS_OK	0	Successful.
UDS_FAILURE	1	General failure, check system logs.
UDS_BAD_PARAM	1000	A bad parameter was passed to a method. This error occurs if a required parameter is missing, the wrong type was passed, or an invalid value was used.
UDS_INTERNAL_ERR	1001	Signals that an internal error occurred. A description is found in the return array and the system logs.

Error Name	Value	Description
UDS_LOCK_ERR	1002	An attempt was made to update an object locked by another user or process. Usually the ID of the contact responsible for locking the object is returned in the return data.
UDS_UPDATE_ERR	1003	An error occurred updating an object. Make sure all required attributes were set and check the system log.
UDS_CREATION_ERR	1004	An error occurred creating a new object. Make sure all required attributes were set and check the system logs.
UDS_NOT_FOUND	1005	A search method failed to find any matches or failed to find an object specified. This can happen if a bad or invalid handle is passed to any method.
UDS_SESSION_TIMEOUT	1006	The current method timed out, the Service Desk server may be heavily loaded or the method itself was bad.
UDS_SERVER_GONE	1007	The Service Desk server connection is lost, UDS methods will no longer function and all list references are lost.

Error Name	Value	Description
UDS_FETCH_ERR	1008	An error occurred while retrieving list data.
UDS_BAD_SESSION	1010	An invalid SID was used.
UDS_CNTXT_TIMEOUT	1011	The SID timed out.
UDS_SECURE_ CHANNEL_REQUIRED	1012	The Web Services (or a web service method) requires a secure channel (for example: SSL) for access, but an unsecured channel is being used. See the Security section.
UDS_SECURITY_ VIOLATION	1013	The attempted operation violates Service Desk security and was aborted.
UDS_OVER_POLICY_ LIMIT	3002	The attempted request is refused because it exceeds the limit defined in the policy.

Chapter 4: Common Elements

This chapter describes Unicenter Service Desk Web Services elements that are shared by Service Desk and Knowledge Tools.

Attribute Data Types

Each attribute of an object is of a specific type that has meaning to the Unicenter Service Desk application, such as string, date, or integer. Knowing the attribute type is essential to correctly retrieving and updating attribute values.

The Service Desk uses an enumeration to identify each data type. These enumeration values are returned in various web methods, as illustrated by the following table:

Data Type	Value
Integer	2001
String	2002
Duration	2003
Date	2004
SREL	2005
UNKNOWN	2006
List (QREL/BREL)	2007
Lrel (many-to-many)	2008
UUID	2009

Integer

The Integer data type represents a 32-bit signed integer. Null an integer attribute by passing the empty string.

String

The String data type represents a character string, where the the maximum length is defined by the database storage allocated for a particular string attribute. For more information about character strings, see "Parsing Tools" in the *Unicenter Service Desk Modification Guide.*

If you attempt to set a string attribute to a value that exceeds its length, the value is truncated and an error message is written to the Service Desk log.

Duration

The Duration data type is an integer representing time duration in seconds. For example, “90” represents one minute and 30 seconds. To set a duration type, use an integer representing the number of seconds for the duration. Negative values are not permitted. To make a duration attribute null, pass the empty string.

Date

The Date data type represents a date value. This is stored as a UNIX-like UTC value in the database (the number of seconds since 1-1-1970). When retrieving date values, the integer UTC is returned. Similarly, use a UTC value to set a date. Negative values are not permitted. Null a date attribute by passing the empty string.

SREL

The SREL data type represents an SREL (Single RELation), which is a pointer to another object. In database terms, it is a foreign key to another table. For example, an Issue object has a pointer attribute to a Contact representing the Assignee.

Most Service Desk Web Services methods permit dot-notation to retrieve information about objects to which an SREL points. For example, to specify the name of a Contact's organization from the context of the Contact, use the following:

```
organization.name
```

You may expand to an arbitrary number of levels as shown in the following example:

```
organization.contact.first_name
```

Dot-notation can only be used to retrieve attribute values, such as using getObjectValues(), or in a Where clause. You cannot use dot-notation to set values.

To set an SREL attribute, such as with updateObject(), always pass the handle of the object to which you want to point. To set an SREL attribute to null, pass the empty string (“”).

List (QREL/BREL)

An object can have a list attribute that represents a one-to-many relationship. These are defined in majic files with the QREL or BREL keywords. A list exists at the object level—it does not take any additional storage in the DBMS.

The Service Desk system handles list collections as abstract data types. The Web Services provides several methods to interact with lists—for references and queries to lists defined in an object, use getRelatedList() and getRelatedListValues(). For more information about lists, see Service Desk Lists in this chapter.

LREL

The LREL data type represents a many-to-many relationship between two object types. An LREL has two names, one for each side of the relationship. The Unicenter Service Desk Web Services provides special functions for interacting with LRELs. For more information about LRELs, see LREL Methods.

UNKNOWN

The Unknown data type represents an unknown data type.

UUID

A UUID is a 128-bit integer (16 bytes) or a 32-byte character that can be used across all computers and networks wherever a unique identifier is required. Such an identifier has a very low probability of being duplicated (for example, a contact ID). UUIDs are used mostly with primary keys.

Lists

Some Web Services methods return *lists*, represented by a unique integer handle. A list is simply a collection of same-type objects. Lists are especially useful when dealing with a large collection of objects (for example, all the contacts in the system) because you can retrieve information about items in a range of the list. The disadvantage is that you must make more method calls to obtain a list handle, retrieve information, and finally, free the list handle. If the expected number of list rows is small, use methods that do not involve list handles, such as doSelect().

The following describes more details about lists:

Lists are homogenous

List may only contain objects of a single type, for example, lists of contacts, list of organizations, and so on.

Lists are Static

For example, if a list object is obtained for all contacts and another contact is added to the system, the update is not reflected in the list. Another list handle must be obtained to get the most current data.

List Handles

A request for a list returns an integer handle representing the list of same-type objects. No other information is sent to the client. The client may query the list for specific information about its rows. When a client is finished with a list, the handle must be released with freeListHandles(). The Unicenter Service Desk server maintains the list, consuming system resources. Therefore, it is important to free lists. Unlike object handles, list handles are not persistent across sessions.

Integer Index

Several methods require an integer index into a list. Lists are zero-based so the first element is at index = 0.

As previously mentioned, using list handles is most useful for larger sets of data that may be queried multiple times. For some operations, however, lists are excessive. Several methods are provided, but the most notable is doSelect(), as it returns requested information about a set of data without the overhead of list handles.

The decision to use list handles versus methods, such as doSelect(), is one of performance and convenience. For example, suppose your application does processing on all 15,000 Contacts in your system. The doSelect() method can retrieve all the contact data in one call, but the reply will be delayed and will negatively impact overall system performance while it assembles and returns a very large data set. The doQuery() method, in this case, will return a list reference very quickly. Ranges of data can be queried from the list to improve response times from the server. A good practice to follow is to use list references if the data set exceeds 250 items.

Sometimes it does not make sense to use list handles. For example, an issue has a list of Activity Logs. Depending on the installation, the number of logs can range from a few to several dozen. It is probably faster to request the data all at once instead of requesting a list reference, querying it for data, and then releasing the list.

Examples of methods that return data sets instead of list references include the following:

- doSelect()
- getRelatedListValues()
- getLrelValues()
- getTaskListValues()
- getValidTaskTransitions()

As previously stated, queries that return a large number of rows can severely impact the performance of the server. To protect against this, Unicenter Service Desk limits the number of rows returned to 250. This affects all Unicenter Service Desk Web Services methods that return lists of objects, including the following:

- doSelect()
- doSelectKD()
- getGroupMemberListValues()
- getListValues()
- getPropertyInfoForCategory()
- getRelatedListValues()
- getTaskListValues()
- getValidTaskTransitions()

This limit applies even if you request one of these methods to retrieve more than 250 rows.

To retrieve large numbers of rows, you should obtain a handle to the list of results and use getListValues() to retrieve chunks of 250 or fewer rows each. This strategy helps keep the server from becoming slow while serving huge amounts of data.

Where Clauses

Several Web Services methods, such as doSelect() and doQuery(), require *where clauses* for searching. A where clause is the string appearing after the 'WHERE' keyword in an SQL statement. For example, a where clause to find contacts (the 'cnt' object) by last name could be the following:

```
last_name = 'Jones'
```

or

```
last_name LIKE 'Jone%'
```

The second example finds all contacts with names beginning with 'Jone', while the first just finds the Jones'.

Unicenter Service Desk supports only a subset of the standard SQL parameters for where clauses, and are listed as follows:

- Logical operators AND, OR, and NOT
- LIKE and IS
- NULL
- IN (see notes following)
- Wildcard characters '%' and '_' for string matches
- All comparison operators: <, >, <=, >=, !=, <>

Parentheses are allowed for grouping. Explicit joins, EXISTS, and GROUP BY elements are not supported.

The column names are simply the object's attribute names. You must use the names defined for the attributes at the object level—do not use the actual DBMS column names (for more information about mapping of object-level names to DBMS column names, see the appendixes "Data Element Dictionary", "Objects and Attributes", and "Table and Object Cross-References" in the *Unicenter Service Desk Modification Guide*). String values must be quoted as in the previous example. Unicenter Service Desk data types, such as date and duration, are treated as integers, as shown in the following example:

```
creation_date > 38473489389
```

Dot-notation is allowed in the where clause to search through SREL (foreign key) types. For example, the following query against the Request ('cr') object, returns all Requests assigned to contacts with a specific last name, as illustrated by the following example:

```
assignee.last_name like 'Martin%'
```

Dot-notation is a great convenience to forming where clauses, but you must be careful to ensure the query is an efficient one. The query in the example could prove inefficient if the contact's last_name attribute was not indexed in the DBMS (however, for this example, it is). To ensure indexes are used to their best advantage when searching through SRELs, make use of the id attributes of the Unicenter Service Desk objects. All tables in Unicenter Service Desk have an index on the id attribute.

The id attribute of an object is easily obtained from the object's handle. As previously outlined, an object's handle is a string of the form "*<objectName>*:*<id>*", where *id* is the value of the id attribute found in every Service Desk object. Simply extract the id portion and use "<attributeName>.id" in the where clause.

An object's id is either of type integer or a UUID. If it is an integer, simply use it as such. For example, to search for Requests with the rootcause pointing to a Root Cause object with handle, "rc:1234", the where clause is:

```
rootcause.id = 1234
```

If the id of an object is a UUID type, you must format it as follows:

```
U'<uuid>'
```

The string representation of a uuid is surrounded by single quotes and prefixed with a capital 'U' character. The string representation of a UUID value is the <id> part of an object handle.

Using the previous example, if you know that the handle for a particular contact is "cnt: 913B485771E1B347968E530276916387", you can form the query, as shown by the following example:

```
assignee.id = U'913B485771E1B347968E530276916387'
```

Do not form where clauses querying the 'persistent_id' attribute, as in the following example:

```
rootcause.persistent_id = 'rc:1234'
```

For more information about handles, see Out-of-the-Box Handles.

IN Clause

The IN clause requires some special explanation. The two syntactic forms are:

```
SREL_attr_name.subq_WHERE_attr[.attr] IN ( value1 [, value2 [,…]] )
SREL_attr_name.[subq_SELECT_attr]LIST_name.subq_WHERE_attr IN (value1, [,value2
[,…]] )
```

The left side of the clause must begin with an SREL-type attribute of the table being queried, which is represented by **SREL_attr_name**. **subq_WHERE_attr** is an attribute of the foreign object, which itself may be another SREL pointer.

For example, a query against the request (‘cr’) object may be coded as follows:

```
category.sym IN (’Soft%’, ’Email’)
```

This translates to the following pseudo-SQL:

```
SELECT … FROM cr WHERE cr.category IN (SELECT persistent_id FROM pcat WHERE sym
LIKE ‘Soft%’ OR sym = ‘Email’)
```

In the previous sub query , ‘pcat’ is the object name pointed to by cr.category.

The second form of the IN clause can search through QREL or BREL lists. For example, to find all requests assigned to an analyst in a specific group, the clause is as follows:

```
assignee.[member]group_list.group IN (U’913B485771E1B347968E530276916387’)
```

The first part of the clause, **assignee**, is an SREL (foreign key) of the cr object, pointing to the cnt object. Next, **group_list**, which is an attribute of the cnt object, is a list of cnt objects that represent groups to which a contact belongs. The last part, **group**, forms the first part of the where clause for the IN sub query. ‘U’913B485771E1B347968E530276916387is the foreign key value to match on **group**. The sub query return is specified by **[member]**. This translates to the following pseudo-SQL statement:

```
SELECT … FROM cr WHERE cr.assignee IN (SELECT member from grpmem WHERE group =
U’913B485771E1B347968E530276916387’)
```

You can specify multiple foreign keys for matching multiple objects by providing a comma-separated list:

```
assignee.[member]group_list.group IN (U’913B485771E1B347968E530276916387’,
U’913B485771E1B347968E530276916300’)
```

You cannot extend the dot notation for this use of the IN clause, for example, the following is not valid:

```
assignee.[member]group_list.group.last_name IN (‘Account Center’)
```

One use of IN is to avoid Cartesian products. For example, the following query results in a Cartesian product and is very inefficient:

```
assignee.last_name LIKE ’MIS%’ OR group.last_name LIKE ’MIS%’
```

Using IN, the query can be coded as follows:

```
assignee.last_name IN ’MIS%’ OR group.last_name IN ’MIS%’
```

This query does not create a Cartesian product; in fact, it creates no joins at all.

Note: The parentheses that normally enclose the list of values on the right side of IN can be omitted if there is only one value in the list. Similarly, you should avoid joins by converting queries, as shown by the following example:

```
assignee.last_name LIKE ‘Smith’
```

to:

```
assignee = U’913B485771E1B347968E530276916387’
```

This avoids the join with some loss in clarity. Using IN, the same partition can be written as follows, with the clarity of the first version and almost the same efficiency as the second version:

```
assignee.last_name IN ‘Smith’
```

The ‘NOT’ keyword cannot be in conjunction with IN, for example, “NOT IN”.

Chapter 5: Web Services Methods

This section provides the details for using the Web Services methods. Each method explains the parameters, description, and returns.

Contact Management Methods

This section explains the Web Services Contact Management methods.

login

The following parameters apply to the login method:

Parameter	Type	Description
username	String	Identifies the user ID.
password	String	Identifies the password.

Description

Login validates a user with Service Desk login validation and returns a unique session ID that is required for most other web method calls. This key should be freed with logout(). A SID may expire if it is not used before a time out elapses.

The username/password is validated exactly the same as the Service Desk Web client; the contact's Access Type specifies the validation method. The default access policy will be applied to control and manage all subsequent accesses after a successful call of this function. The login user (not proxy contact specified in the default policy) is then responsible for subsequent web service activities. All function group security and data partition are enforced for the login user.

Returns

The following is returned:

Parameter	Type	Description
<SID>	Integer	Identifies the unique SID to use for all other Web Services calls.

loginService

The following parameters apply to the loginService method:

Parameter	Type	Description
username	String	Identifies the user ID.
password	String	Identifies the password.
policy	String	(Required) Identifies the policy code, which must be in plain text. Although it is required, it may be empty.

Description

Lets users log in with a conventional username/password authentication scheme where if valid, the system returns a unique session ID. This key should be freed with logout(). A SID may expire if it is not used before a timeout elapses.

User authentication is performed on the username and password while access control is applied based on the policy specified. The authentication is performed as described in login(). Empty policy will allow default policy to be applied automatically. The login user (not the proxy contact specified in the policy) is responsible for subsequent web service activities. All function group security and data partitions are enforced for the login user.

Returns

The following is returned:

Parameter	Type	Description
SID	Integer	Specifies the unique SID to use for all other Web Services calls.

loginServiceManaged

The following parameters apply to the loginServiceManaged method:

Parameter	Type	Description
policy	String	(Required) Identifies the policy, which must be must be in plain text.
encrypted_policy	String	(Required) Identifies the digital signature of the policy code, encrypted with the policyholder’s private key. It is in BASE64 text format.

Description

Performs the user authentication by locating the policy through the plain text policy code, finding the policyholder’s public key associated with the policy, decrypting the encrypted policy code, matching decrypted content with the policy code, and finally, opening a session with a back-end server. The returned session ID (SID) can be used for subsequent web services method invocations. Proxy contact specified in the policy is responsible for all subsequent web service activities initiated. All function group security and data partition are enforced for the proxy contact defined in the policy.

It is also important to note that the encrypted_policy parameter is in the BASE64 text format, and it is necessary to perform proper conversion from the binary format. The SID should be freed with logout(). A SID may expire if it is not used before a timeout elapses.

Returns

The following is returned:

Parameter	Type	Description
SID	String	Identifies the unique session ID (SID) to use for all other Web Services calls. It is in plain text format.

impersonate

The following parameters apply to the impersonate method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
username	String	(Required) Identifies the user name of the user being impersonated.

Description

Lets an administrator switch the user responsible for all web services activities in a current web services session without additional user authentication. Invoking this method is allowed only if the current web services session is started by using the PKI access authentication scheme and the access policy is defined to allow impersonation.

The Access Type of the user to be impersonated is checked against the Access Type of the proxy user of the policy used in the current web services session. If the access_level of the new user's access type is less than or equal to the grant_level of the proxy user's access type, this method will replace the current user with the new user. A new web services session starts while the old session ends. A new SID is then returned. In addition, the new user is given the responsibility for all subsequent activities initiated in this new session. The function group security and data partition are enforced for the new user.

Returns

The following is returned:

Parameter	Type	Description
SID	Integer	Identifies the unique SID to use for all other Web Services calls.

logout

The following parameter applies to the logout method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.

Description

Invalidates a SID obtained from login(), loginService(), and loginServiceManaged().

Returns

Nothing.

getHandleForUserid

The following parameters apply to the getHandleForUserid method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
userID	String	Identifies the user ID upon which to query.

Description

Returns the persistent handle for a Contact represented by userID.

Returns

The following is returned:

Parameter	Type	Description
<Handle>	String	Identifies the contact's handle.

getAccessTypeForContact

The following parameters apply to the getAccessTypeForContact method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
ContactHandle	String	Identifies the contact handle upon which to query.

Description

Returns a handle for the Access Type for a Contact.

Every Contact is assigned an Access Type object, which defines a Contact's permissions and security. Note the access_type field on a Contact is not required. To accommodate this, a single Access Type is marked as the default for Contacts who do not have an object specifically assigned.

This method returns the Access Type directly assigned to a Contact (that is, the information to which the access_type field points in the Contact record) or it returns the default. Typical value methods, such as getObjectValues() or getListValues(), may not return the correct Access type; therefore, these are not the accurate methods for retrieving the Contact's Access.

Returns

A string handle for an Access Type object.

getContact

The following parameter applies to the getContact method:

Parameter	Type	Description
SID	String	Identifies the session retrieved from logging in.
contactId	String	Identifies the unique ID of the contact to retrieve. contactId is UUID in string format.

Description

Retrieves information on all contacts.

Returns

A <UDSObject> node with a <UDSObject> node describing a contact with some of the following child <Attributes> nodes:

XML Element Value	Type	Description
contactid	String	Specifies a unique ID of the contact. contactId is UUID in string format.
userid	String	Indicates the user name of the contact.
last_name	String	Indicates the last name of the contact.
first_name	String	Specifies the first name of the contact.
middle_name	String	Specifies the middle name of the contact.
location	String	Indicates the location of the contact.
dept	String	Identifies the department of the contact.
organization	String	Identifies the organization of the contact.
email_address	String	Identifies the email address of the contact.
pemail_address	String	Signifies the alternate email address of the contact.
phone_number	String	Indicates the phone number of the contact.

XML Element Value	Type	Description
alt_phone	String	Indicates the alternate phone number of the contact.
address1	String	Specifies the address of the contact.
address2	String	Specifies the alternate address of the contact.
city	String	Identifies the city of the contact.
state	String	Identifies the state of the contact.
zip	String	Indicates the zip code of the contact.
country	String	Indicates the country of the contact.
delete_flag	Integer	Indicates whether the contact is active: 0=Active 1=Inactive

createTicket

The following parameters apply to the createTicket method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
Description	String	(Optional) Identifies the description of the ticket.

Parameter	Type	Description
problem_type	String	Specifies the code (not the persistent ID) for an existing problem type for the policy under which a Web Services application is running. If this is blank or contains a bad value, the default problem type is used.
Userid	String	(Optional) Specifies the user ID of the end user for the new ticket. If this is blank or the supplied user ID is not found, the proxy contact defined in the access policy is used and the ticket is still created.
Asset	String	(Optional) Identifies the handle of an asset to be attached to the ticket.

Parameter	Type	Description
DuplicationID	String	(Optional) Allows callers to assist the duplication handling routines used for classifying tickets as being unique or different. If duplicate handling is on, this string is inspected after other duplicate handling criteria match to determine whether this is a unique or duplicate call to this method. For more information about Duplicate Ticket Handling, see Duplicate Ticket Handling.

Description

Creates a ticket based on the rules defined in both the Service Aware Policy and the specified Problem Type. The Problem Type specifies the type of ticket created by a Request/Change/Issue template. The supplied description is copied in and the user is set from the access policy's proxy contact.

If input problem_type does not exist, the policy's default problem type is used. The problem type also defines how to handle duplicate ticket creation, and additional returned data. For detailed information about configuring access policy and problem types, see the chapter "External Specifications".

Returns

Returns the new ticket's handle, ticket number, and the user-specified return data defined in the problem type that is used to create the ticket.

XML Element	Type	Description
<UDSObject>	N/A	Returns a partial set of attributes because it is a high-level method designed to simplify the process. Their values come from the following methods, and the XML element is returned so it is consistent with the returns of these methods: ■ createRequest() ■ createChange() ■ createIssue() The body of this tag is empty.
NewTicketHandle	String	Identifies the new ticket's handle.
NewTicketNumber	String	Identifies the new ticket's number (its "ref_num" or "chg_ref_num" attribute).

XML Element	Type	Description
ReturnAppCode	String	Identifies the user-specified data for the problem type intended for use within the application code, especially for actions. You can set this value in the Application Data Return field on the Returned Data tab of the Problem Type Detail window.
ReturnUserCode	String	Identifies the user-specified data for the problem type intended for display to the end user, or for log entries. You can set this value in the User Data Return field on the Returned Data tab of the Problem Type Detail window.

findContacts

The following parameters apply to the findContacts method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
userName	String	Returns only the contacts with this user name. You can do wildcard searches by using the percent (%) sign. For example, to search for all contacts where the User Name begins with Smi, specify Smi%.
lastName	String	Returns only the contacts with this last name. You can do wildcard searches by using the percent (%) sign.

Parameter	Type	Description
firstName	String	Returns only the contacts with this first name. You can do wildcard searches by using the percent (%) sign.
email	String	Returns only the contacts with this email address. You can do wildcard searches by using the percent (%) sign.
accessType	String	Returns only the contacts with this access type. You can specify multiple roles by separating them with commas. Specify the following: ▪ 2402 for Administrators ▪ 2403 for Analysts ▪ 2404 for External Customers ▪ 2405 for Internal Customers ▪ 2406 for Knowledge Engineers ▪ 2407 for Knowledge Managers
inactiveFlag	Integer	Returns only the contacts that are inactive or active. Specify the following: ▪ 0 for active ▪ 1 for inactive ▪ –999 for all

Description

Retrieves the list of contacts.

Returns

A <UDSObjectList> node with zero or more <UDSObject> nodes describing contacts with the following <Attributes> child nodes:

XML Element Value	Type	Description
id	UUID	Specifies the unique ID of the contact.
userid	String	Specifies the user name of the contact.
last_name	String	Identifies the last name of the contact.
first_name	String	Identifies the first name of the contact.
access_type	Integer	Specifies the Role ID of the contact.
delete_flag	Integer	Indicates whether the contact is active or inactive: ▪ 0 for active ▪ 1 for inactive ▪ –999 for all

getPermissionGroups

The following parameters apply to the getPermissionGroups method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
groupId	Integer	Returns only the group with this ID. Pass zero (0) when you do not want to use this parameter.

Description

Retrieves the list of Permission Groups.

Returns

A <UDSObject> node with zero or more <UDSObject> nodes describing Permission Group with the following <Attributes> child nodes:

XML Element Value	Type	Description
id	Integer	Identifies the unique ID of the group.
GRP_LIST_KEY	String	Shows a list of the IDs of Service Desk groups, separated by commas.
GRP_LIST	String	Displays a field containing the entire group list.

Group Management Methods

This section explains the Web Services Group Management Methods.

addMemberToGroup

The following parameters apply to the addMemberToGroup method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
ContactHandle	String	Identifies the handle of the contact to add as a member.
GroupHandle	String	Identifies the group to which you will add the contact.

Description

Adds a contact to a group. Nothing happens if the contact is already a member.

Returns

Nothing.

removeMemberFromGroup

The following parameters apply to the remove MemberFromGroup method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
ContactHandle	String	Identifies the handle of the contact to remove.
GroupHandle	String	Identifies the group from which you will remove the contact.

Description

Removes a contact from a group. Nothing happens if the contact is not a member.

Returns

Nothing.

getGroupMemberListValues

The following parameters apply to the getGroupMemberListValues method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
whereClause	String	Identifies the SQL where clause.
numToFetch	Integer	Determines the maximum number of records to return. This cannot be zero (0). Use ‘1’ to return all.
attributes	String[]	Identifies the array of attribute names for which to retrieve values (see the following description).

Description

Functions similarly to getListValues(), except that it queries the system’s group and member relationships. The system uses a special ‘Group Member’ (grpmem) object for each group/member relationship. The Unicenter Service Desk system administers grpmem objects behind the scenes (you do not manipulate them directly), and they are essential for certain queries.

The grpmem object simply contains two pointers, one to a Contact and the other to a Group, which in itself is a Contact object. The attribute names are ‘member’ and ‘group’, respectively. Because these are pointers, you must use dot-notation to form a query as normal. For example, to find all Contacts with a last name beginning with ‘B’ and in a group with the name, “Seattle”, you would use the following:

```
member.last_name LIKE ‘B%’ AND group.last_name = ‘Seattle’
```

You may also use handles as normal, illustrated by the following:

```
member.last_name LIKE ‘A%’ AND group.id = U‘555A043EDDB36D4F97524F2496B35E75’
```

It is important to note that this method can retrieve values from all members and groups, not just a single group. To simply get information about all the members of a specific group or member, just specify a handle in the where clause. For example, the following would retrieve values from a specific group:

```
group.id = U‘555A043EDDB36D4F97524F2496B35E75’
```

The important concepts to remember are:

- grpmem object contains two pointer attributes, one is to member and the other is to group
- grpmem object exists for each group/member relationship

The grpmem method actually queries the grpmem object table, thereby returning an object representing a relationship between two contacts. Therefore, the attribute values you want to fetch in attributes must use dot-notation from the grpmem object. To fetch values from the member, all your attribute names should be of the form, ‘member.*ATTRNAME*’, as illustrated by the following example:

```
‘member.last_name’
```

To fetch values from the group, use ‘group.*ATTRNAME*’.

Note: For an example of efficient querying of groups and members, see Where Clauses.

Returns

Automatically returns no handles. The <Handle> element in the return is always empty. To request the member or group handle for each row, use one of the following in the attributes parameter described in the table.

- “member.persistent_id”
- “group.persistent_id”

XML Element	Type	Description
<UDSObjectList>	N/A	Identifies the outer element, which contains an array of <UDSObject> elements. Each object is really a grpmem object. For information about the contents of each <UDSObject> element, see XML Object Returns.

Business Methods

This section describes the Web Services Business methods.

createIssue

The following parameters apply to the createIssue method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
creatorHandle	String	Identifies the handle of the contact responsible for the creation of the issue (the log agent). Pass an empty string to specify the default Service Desk user.
attrVals	String[]	Identifies the array of name-value pairs that is used to set the initial attribute values for the new issue.
propertyValues	String[]	(Optional) Identifies the array of values for any properties that are attached to the new issue.
template	String	(Optional) Identifies the handle of an issue template (iss_tpl object) from which to create the new issue.

Parameter	Type	Description
attributes	String[]	Identifies the sequence of attribute names from the new object for which to return values. Dot-notation is permitted. If this field is empty, all attribute values are returned.

Description

Creates a Service Desk Issue (iss) object. For more information about creating an Issue object with properties, see createRequest().

Note: You *must* use this function to create a new Issue; do not use createObject().

Returns

Returns the new object's handle, along with *all* of its attribute values. List and LREL types are also returned, but as empty elements. For more information about the return format, see the XML Object Returns.

XML Element	Type	Description
<UDSObject>	N/A	Identifies the standard UDSObject element containing the handle and requested attribute values.
<NewIssueHandle>	String	Identifies the new issue's handle.
<NewIssueNumber>	String	Identifies the new issue's number (its "ref_num" attribute).

createQuickTicket

The following parameters apply to the createQuickTicket method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
customerHandle	String	Identifies the customer handle used to create the ticket.
description	String	(Optional) Identifies the description of the ticket.

Description

Creates a ticket based on the preferred document type of the user represented by customerHandle. Every contact's access rights are determined by an Access Type record, which also sets the contact's preferred document type (Request, Incident, Problem, Issue or Change Order). If a contact's document type is Issue, this method will create an Issue; if the document type is Request, a Request is created, and so on. The contact represented by customerHandle is used to set the end user/customer field of the new ticket. The ticket's description is set to the input *description* value.

Returns

Returns the new ticket’s handle, ticket number, and a brief representation of the new ticket in <UDSObject> format.

XML Element	Type	Description
<UDSObject>	N/A	Returns a partial set of attributes because it is a high-level method designed to simplify the process. Their values come from the following methods, and the XML element is returned so it is consistent with the returns of these methods: ▪ createRequest() ▪ createChange() ▪ createIssue()
NewTicketHandle	String	Identifies the new ticket’s handle.
NewTicketNumber	String	Identifies the new ticket’s number (its “ref_num” or “chg_ref_num” attribute).

closeTicket

The following parameters apply to the closeTicket method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
description	String	(Optional) Identifies the description of the ticket, which can be used in the Close activity log.
ticketHandle	String	Identifies the ticket to close.

Description

Sets the status of the ticket to “Closed” and adds an activity log with the input description.

Returns

The handle to the activity log object created. It provides the same return as the changeStatus() method.

XML Element	Type	Description
<LogHandle>	String	Specifies the handle for new activity log (0 or more of these can be returned).

createRequest

The following parameters apply to the createRequest method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
creatorHandle	String	Identifies the handle of the contact responsible for the creation of the request (the log agent). Pass an empty string to specify the default Service Desk user.
attrVals	String[]	Identifies an array of name-value pairs that is used to set the initial attribute values for the new request.
template	String	(Optional) Identifies the handle of the request template (cr_tpl) from which to create the new request.
propertyValues	String[]	(Optional) Identifies the array of values for any properties that are attached to the new request.
attributes	String[]	Specifies the sequence of attribute names from the new object for which to return values. Dot-notation is permitted. If this field is empty, all value-based attribute values are returned.

Description

Creates a Service Desk Request (cr) object. You *must* use this function to create a new Request; do not use createObject().

propertyValues is a list of values for each Property object that will be attached to the new Request. The Properties that are attached are determined by the new Request's 'category' attribute value. All properties created from the Service Desk Web Services interface will have a default value (for more information, see Categories and Properties), which is important because a Request will not save until all of its Properties marked "required" have a value.

You may override the default by supplying values for any properties that will be attached when the Request is created. You must supply this information before the Request is created, since createRequest() attempts to back-store the object you most recently create. Use getPropertyInfoForCategory() to get a list of properties for a specific Category. This function returns the properties in order of their 'sequence' attribute, which is the expected order of the *propertyValues* array. For example, if the sequences and symbols of the properties are as follows:

```
100 – Hard Drive Size

200 – CPU

300 – Memory
```

The *propertyValues* array, depending on the programming language, may appear as follows:

```
[“40 GB”, “Pentium 4 1.7 Ghz”, “256”]
```

getPropertyInfoForCategory() indicates which Properties are marked *required*.

If you do not set the Request category or do not want to set any Property values, pass an empty string for *propertyValues.*

If you do not want to rely on the default property values, the following is the suggested order for creating a new Request (or Issue or Change Order):

1. Retrieve a list of Categories/Areas. The object name for Request Area is 'pcat'.
2. Call getPropertyInfoForCategory() and examine the list of properties for the category of the new Request/Issue/Change.
3. Create a value array for each of the properties returned. This is identified by the *propertyValues* parameter for the create operation.
4. Assemble the *attrVals* array and call the create method.

As an alternative to the previous procedure, you can retrieve the list of properties using getRelatedListValues() after createRequest() returns. Properties are stored in the ‘properties’ list for a Request.

Depending upon the application, it may be faster to at least cache the list of Categories, since this data does not change often at many client sites.

Note: By default, this method creates a Request. If you are using the ITIL methodology, you need to set the ‘type’ attribute in the attrVals array to define whether you are creating an Incident or a Problem ticket. For more information about ITIL procedures, see Note on Using the ITIL Methodology Installation.

Returns

Returns the new object’s handle with *all* of its attribute values. List and LREL types are also returned, but as empty elements. For more information about the return format, see XML Object Returns.

XML Element	Type	Description
<UDSObject>	N/A	Identifies the standard UDSObject element containing the handle and requested attribute values.
<NewRequestHandle>	String	Identifies the new request’s handle.
<NewRequestNumber>	String	Identifies the new request’s number (its “ref_num” attribute).

createChangeOrder

The following parameters apply to the createChangeOrder method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
CreatorHandle	String	Identifies the handle of the contact responsible for the creation of the change order (the log agent). Pass an empty string to specify the default Service Desk user.
attrVals	String[]	Identifies an array of name-value pairs that is used to set the initial attribute values for the change order.
template	String	(Optional) Identifies the handle of the change template (chg_tpl object) from which to create the new issue.

Parameter	Type	Description
propertyValues	String[]	(Optional) Identifies the array of values for any properties that are attached to the new change order.
attributes	String[]	Specifies the sequence of attribute names from the new object for which to return values. Dot-notation is permitted. If this field is empty, all value-based attribute values are returned.

Description

Creates a Service Desk Change Order (chg) object. You *must* use this function to create a new change order; do not use createObject(). For more information about creating a Change Order object with properties, see createRequest().

Returns

The new object's handle, along with *all* of its attribute values. List and LREL types are also returned, but as empty elements. For more information about the return format, see XML Object Returns.

XML Element	Type	Description
<UDSObject>	N/A	Identifies the standard UDSObject element containing the handle and request attribute values.
<NewChangeHandle>	String	Identifies the new change order's handle.
<NewChangeNumber>	String	Identifies the new change order's number (its 'chg_ref_num' attribute).

getPropertyInfoForCategory

The following parameters apply to the getPropertyInfoForCategory method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
categoryHandle	String	Identifies a category handle.
attributes	String[]	Identifies the names of one or more attributes from the property template object for which to fetch values. If this is empty, all value-based attributes are fetched.

Description

Information about Properties for the specified category. This method is used to help pre-populate Request/Issue/Change Order properties on insert operations with user-defined data.

Depending on the category, this method queries either the ‘prptpl’ or the ‘cr_prptpl’ object types. Both types are nearly identical. The suggested attributes to fetch are ‘sequence’, ‘label’, ‘description’ and ‘required’.

Returns

A <UDSObject> element containing a sequence of <UDSObject> elements, in the order of the ‘sequence’ attribute.

transfer

The following parameters apply to the transfer method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
creator	String	Identifies the handle of the contact responsible for the activity.
objectHandle	String	Identifies the handle for a request, issue, or change order. Any other object type is rejected.
description	String	Identifies the description for the activity, which will appear in the activity log.
setAssignee	Boolean	Used to update the assignee field with the value in newAssignee, if the value is true.
newAssigneeHandle	String	Identifies the new assignee for the object.
setGroup	Boolean	Updates the group field, if true.
newGroupHandle	String	Identifies the new group for the object.

Parameter	Type	Description
setOrganization	Boolean	Update the organization, if the value is true.
newOrgnizationHandle	String	(Issues and Change Orders only) Identifies the new organization for the object.

Description

Performs a transfer activity on an Issue, Request or Change Order. This method corresponds to the “Activities—Transfer” command in the Service Desk interface. This method generates an activity log and optionally sets a new assignee, group, or organization.

The assignee, group or organization is not updated unless one or more of the corresponding *setAssignee/setGroup/setOrganization* parameters is set to true.

If the companion parameter is false, then transfer will not attempt to update the field, even if a value is passed for that field. For example, if *setAssignee* is passed as false, transfer will not update the assignee even if *newAssignee* specifies a value. If the *setXXXX* parameter is true, then the field is updated. Pass the empty string to set a field to empty/null.

Returns

One or more handles to the activity log objects created. The returns are under a parent element named <Logs>.

XML Element	Type	Description
<LogHandle>	String	Identifies the handle for the new activity log (zero or more of these can be returned).

escalate

The following parameters apply to the escalate method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
creator	String	Identifies the handle of the contact responsible for the activity.
objectHandle	String	Identifies the handle for a request, issue, or change order. Any other object type is rejected.
description	String	Identifies the description for the activity, which will appear in the activity log.
setAssignee	Boolean	Updates the assignee field, if true.
newAssigneeHandle	String	Identifies the handle of the new assignee for the object.
setGroup	Boolean	Updates the group field with the value in newGroupHandle, if true.
newGroupHandle	String	Identifies the handle of the new group for the object.
setOrganization	Boolean	Sets the organization field with the value specified in the newOrganizationHandle, if true.
newOrgnizationHandle	String	(Issues and Change Orders only) Identifies the handle of the new organization for the object.

Parameter	Type	Description
setPriorityHandle	Boolean	Updates the priority field with the value specified in newPriority, if true.
newPriority	String	Identifies the handle of the new priority for the object.

Description

Performs an escalate activity on an Issue, Request or Change Order. This method generates an activity log and optionally sets a new assignee, group, priority and/or organization.

It corresponds to the “Activities—Escalate” command in the Service Desk interface.

The assignee, group, or organization is not updated unless one or more of the corresponding *setAssignee/setGroup/setOrganization* parameters is set to true. If the companion parameter is false, then escalate will not attempt to update the field, even if a value is passed for that field. For example, if *setAssignee* is passed as false, escalate will not update the assignee even if *newAssignee* specifies a value. If the *setXXXX* parameter is true, then the field is updated. Pass the empty string to set a field to empty/null.

Note: Organization is not used for Requests.

Returns

One or more handles to the activity log objects created.

XML Element	Type	Description
<LogHandle>	String	Identifies the handle for the new activity log (zero or more of these can be returned).

changeStatus

The following parameters apply to the changeStatus method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
creator	String	Identifies the handle of the contact responsible for the activity.
objectHandle	String	Identifies the handle for a request, issue, or change order. Any other object type is rejected.
description	String	Identifies the description for the activity, which will appear in the activity log.
setStatus	String	(Optional) Identifies the handle of the new status for the object
newStatusHandle	String	Identifies the handle of the status for the object.

Description

Performs a status change activity on an issue, request, or change order. This method generates an activity log and optionally sets the status value. It corresponds to the “Activites—Update Status” command in the Service Desk interface.

Returns

The handle to the activity log object created.

XML Element	Type	Description
<LogHandle>	String	Identifies the handle for the new activity log (zero or more of these can be returned).

createActivityLog

The following parameters apply to the createActivityLog method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
creator	String	Identifies the handle of the contact responsible for the activity.
objectHandle	String	Identifies the handle for a request, issue, or change order. Any other object type is rejected.
description	String	Identifies the description for the activity, which will appear in the activity log.

Parameter	Type	Description
LogType	String	Identifies the type of log to create—see the following Description.
TimeSpent	Integer	Sets the Time Spent field for the activity log, which is the duration of the activity. Pass zero for the default.
Internal	Boolean	Identifies the values that apply: ▪ True = Internal-only activity ▪ False = Non-internal activity that can be viewed by everyone.

Description

Creates an activity log for a specified request, issue or change order. This method corresponds to, “Activities – Log Comment/Research/Callback” on a Change/Request/Issue detail in the Service Desk interface. *LogType* is the code attribute for the activity type of the new log. The most common codes are as follows:

- “CB” (Callback)
- “RS” (Research)
- “LOG” (Log Comment)

For more information about all possible values, see the Act_Type (aty) table in the appendix “Data Element Dictionary” of the *Unicenter Service Desk Modification Guide*. The Service Desk Administrative Client also shows the code values. To access the code values, select from the Main Menu Administration, then select Notification, Activity Notifications.

Returns

The handle to the activity log object created.

XML Element	Type	Description
<LogHandle>	String	Identifies the handle for the new activity log.

attachChangeToRequest

The following parameters apply to the attachChangeToRequest method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
creator	String	Identifies the handle of the contact responsible for the activity.
requestHandle	String	Identifies the request to which you attach the change.

Parameter	Type	Description
changeHandle	String	Identifies the handle of the change to add. If this is blank, a new change order is created. For additional information, see the following Description.
changeAttrVals	String[]	Identifies the attribute-value pairs used to initialize a new change order if the changeHandle is blank. For more information, see the following Description.
description	String	(Optional) Identifies the description of the activity.

Description

Attaches a new or existing change order to a request. It corresponds with "Activities—New Change Request" or "Activities—Attach to Existing Request" on a Request detail in the Service Desk interface.

To create a new change order, pass the empty string in *changeHandle*. The system will create a new change order with values initialized from the request, including the change's Requestor, Affected End User, Description, and Priority (you can see this effect in the Service Desk interface). You can override these or set additional values with *changeAttrVals,* which is a name-value array similar to what is passed for *createObject().*

To attach an existing change order, specify a handle in the *changeHandle* parameter. In this case, *changeAttrVals* is ignored.

If a new change order is created, *description* is used on the new change order's activity log. If an existing change is attached, *description* is used on the request's activity log.

If the request already has an attached change order, the following error is returned:

```
UDS_CREATION_ERROR
```

Note: This method works exactly the same for the ITIL methodology—simply verify that you are passing the appropriate handle of an ITIL Incident or Problem to the method.

Returns

The following is returned:

XML Element	Type	Description
<changeHandle>	String	Identifies the handle for the change order, created or attached.

detachChangeFromRequest

The following parameters apply to the detachChangeFromRequest method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
creator	String	Identifies the handle of the contact responsible for the activity.
requestHandle	String	Identifies the handle of a request.
Description	String	(Optional) Identifies the description of the activity.

Description

Detaches a change order from a request. This method corresponds with “Activities—Detach Change Order” on the Service Desk client. The change order is not deleted from the system.

There is no effect if the request did not have an attached change.

Returns

The handle of the request’s activity log marking the event.

logComment

The following parameters apply to the logComment method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
ticketHandle	String	Identifies the handle of the ticket where the activity log should be added.

Parameter	**Type**	**Description**
Comment	String	Identifies the comment text.
internal_flag	String	Identifies the internal flag. Set to True if the new activity log should be marked as internal.

Description

Attaches a ‘Log Comment’ activity log to a ticket. It is a simplified version of createActivityLog().

Returns

Nothing.

notifyContacts

The following parameters apply to the notifyContacts method:

Parameter	**Type**	**Description**
SID	Integer	Identifies the session retrieved from logging in.
creator	String	Identifies the handle of the contact responsible for the activity.

Parameter	Type	Description
contextObject	String	Identifies the handle for a request, issue, or change order. The following applies: ▪ It is the context for the notification. Any other object type is rejected. ▪ An activity log is added to the object to record the notification.
messageTitle	String	Identifies the title of the notification message.
messageBody	String	Identifies the body of the notification message.
notifyLevel	Integer	Indicates the notification level. Specify an integer from 1 (Low) to 4 (Emergency).
notifyees	String[]	Identifies the array of contact handles to notify.
Internal	Boolean	Indicates internal-only notification. Set to True to flag for an internal-only notification, which guarantees that the message is sent only to those who can view internal logs, and the resulting activity log is flagged as internal.

Description

Sends a notification to one or more contacts. This is equivalent to the Manual Notify activity on requests, issues, and change orders.

Returns

The following is returned:

XML Element	Type	Description
<LogHandle>	String	Identifies the handle for new activity log.

clearNotification

The following parameters apply to the clearNotification method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
lrObject	String	Identifies the handle to a Notify Log Header (lr) object.
clearBy	String	Identifies the name of the contact responsible for the clear operation.

Description

Clears a notification message.

Returns

The new status of the notification message.

getPolicyInfo

The following parameters apply to the getPolicyInfo method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.

Description

Returns information about the access policy that is controlling and managing the current Service Desk Web Services session.

Returns

The following XML string:

XML Element	Type	Description
<SAPolicy>	N/A	Identifies the detailed information of this access policy and its related problem types.

The content of <SAPolicy> is shown as follows:

```
<SAPolicy>
<Name> name of policy </Name>
<Code> policy code </Code>
<ContactName> policy proxy contact's combo name </ContactName>
<ContactHandle> handle of policy's contact </ContactHandle>
<Access>
        <TicketCreation> limitation </TicketCreation>
        <ObjectCreation> limitation </ObjectCreation>
        <ObjectUpdate> limitation </ObjectUpdate>
        <Attachments> limitation </Attachments>
        <Queries> limitation </Queries>
        <Knowledge> limitation </Knowledge>
</Access>
<ProblemTypes> (zero or more <ProblemType> elements)
        <ProblemType>
                <Code>code of a problem type </Code>
                <Status>active or inactive </Status>
        </ProblemType>
</ProblemTypes>
</SAPolicy>
```

getTaskListValues

The following parameters apply to the getTaskListValues method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
objectHandle	String	Identifies the object handle for an issue or change order.
attributes	String[]	Identifies the sequence of attribute names for which to fetch values. Dotted-attributes are permitted. If this is blank, all attributes are fetched.

Description

Returns values for all the tasks associated with the specified issue or change order.

Note: This is a convenience method. The same list could be obtained using doSelect().

Returns

The following:

XML Element	Type	Description
<UDSObjectList>	N/A	Outer element—contains zero or more <UDSObject> elements as described in XML Object Returns. Each <UDSObject> node represents a Task. The nodes are ordered by the Task's 'sequence' attribute. For information about the contents of each <UDSObject> element, see XML Object Returns.

getValidTaskTransitions

The following parameters apply to the getValidTaskTransitions method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
taskHandle	String	Identifies the handle to a Workflow task (for an issue or change order).

Parameter	Type	Description
attributes	String[]	Identifies the names of attributes to fetch from the 'tskstat' object. If this field is empty, all value-based attributes are returned.

Description

Returns all of the possible values for the 'status' attribute of a particular task. The Status codes to which a task may be set depend upon several factors, such as the current status of the task, and restrictions set by the administrator.

Note: *taskHandle* can be a task owned by either a change order or an issue. The objects returned are Task Status ('tskstat') objects used for both types of tasks.

Returns zero or more Status objects to which a task can be set.

Returns

The following:

XML Element	Type	Description
<UDSObjectList>	N/A	Identifies the outer element, which contains zero or more <UDSObject> elements with the requested attribute values. For information about the contents of each <UDSObject> element, see XML Object Returns.

Asset Management Methods

It is possible for a client site to enhance the asset object using *extensions*. Asset extensions are separate tables that hold extra attribute information for an asset. The extension table is linked to a particular asset by using the asset ID as a foreign key. Unicenter Service Desk Web Services ships two predefined extensions, Computer and Software. For more information, read the text executed by the following command:

```
/bopcfg/majic/assetx.maj
```

The asset’s family attribute determines if the asset has an extension. Setting the Class attribute generally sets the family at asset creation time. To determine if an asset has an extension, query the ‘extension_name’ attribute of the asset’s family (for example, “family.extension_name”).

To retrieve values from an extension object, query it like any other object by using the following method:

```
getObjectValues())
```

To get the handle for a particular extension object, use the following method:

```
getAssetExtensionInformation()
```

Update an extension object like you would any other object by using the following method:

```
updateObject()
```

We do not recommend that you create your own extension objects. One is created for you, if needed, when createAsset() is called. Because of this automatic creation, we recommend that you only use createAsset() to create asset objects.

Note: If you are using the ITIL methodology, remember that Asset and Configuration Item are interchangeable in this context. For more information, see Note on Using the ITIL Methodology Installation.

createAsset

The following parameters apply to the createAsset method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
attrVals	String[]	Identifies the array of name-value pairs used to set the initial attribute values for the new asset.
attributes	String[]	Identifies the sequence of attribute names from the new object for which to return values. Dot-notation is permitted. If this field is empty, all attribute values are returned.

Description

Describes the recommend method for creating a new asset. If you intend to create an asset with an extension, be sure to set the Asset Class attribute in the *attrVals* section.

Note: If you are using the ITIL methodology, use this method to create a Configuration Item. See the Note on Using the ITIL Methodology Installation.

Returns

A <UDSObject> element containing the new object's handle, along with attribute values specified in the *attributes* parameter. If the *attributes* parameter is empty, *all* attribute values are returned. List and LREL types are also returned, but as empty elements. For more information about the return format, see XML Object Returns.

XML Element	Type	Description
<UDSObject>	N/A	Identifies the standard <UDSObject> element containing the handle and requested attribute values.
<NewAssetHandle>	String	Identifies the new request's handle.
<ExtensionHandle>	String	Identifies the handle for the new Asset's extension. If no extension was created, this field is empty.
<ExtensionName>	String	Identifies the name for the new asset's extension. If no extension was created, this field is empty.

getAssetExtensionInformation

The following parameters apply to the getAssetExtensionInformation method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
assetHandle	String	Identifies the asset to query.
attributes	String[]	Identifies the standard array of attributes from the asset extension object from which to request values. If this value is empty, all attributes are returned.

Description

Returns extension information for an asset. If the asset does not have an extension, nothing is returned.

An asset has an extension if a value exists for its “family.extension_name” property. This property is empty if the asset does not have an extension.

Returns

The following elements with empty values if the asset does not have an extension:

XML Element	Type	Description
<UDSObject>	String	Identifies all the attribute values for the extension. For information about the contents of each <UDSObject> element, see XML Object Returns.
<ExtensionHandle>	String	Identifies the extension’s handle.
<ExtensionName>	String	Identifies the name for the asset’s extension.

addAssetLog

The following parameters apply to the addAssetLog method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
assetHandle	String	Identifies the asset handle.
contactHandle	String	(Required) Identifies the handle of the contact used for the log’s author.
logText	String	Identifies the text for the new asset log.

Description

Adds a new log entry for an asset. The log's author is the user associated with the SID.

Returns

Nothing.

createAssetParentChildRelationship

The following parameters apply to the createAssetParentChildRelationship method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
parentHandle	String	Identifies the asset handle for the parent.
childHandle	String	Identifies the asset handle for the child.

Description

Makes *assetParent* a parent of *assetHandle*. Web Services creates a separate object (the *hier* object, which is the *Assignment* table) for parent-child relationships between assets. These are stored in related lists, *child_hier* and *parent_hier,* in the Asset (nr) object.

Returns

Handle of the new hier (Assignment) object.

List/Query Methods

Two paradigms are available for working with lists. One paradigm uses a list handle for referring to and making queries on a server-side list and the other simply performs a SQL-like select.

If you need to maintain reference to a static list, use the methods that return list handles. These methods are especially useful when working with very large lists. For example, your application may need to perform operations using the entire table of 10,000 Contacts. Downloading values for all 10,000 at once could result in an unacceptable performance lag (this condition is actually prevented by the system—see Where Clauses). With a list handle, however, you can select a range of rows upon which to query.

The primary drawback to using a list handle is the extra method calls it requires. At least two or three calls are necessary, as indicated by the following:

- One to get the handle
- A second (or third) to retrieve values
- A final call to free the list

You need to balance the amount of remote method calls versus the expected amount of data returned.

Note: Unicenter Service Desk restricts the amount of data that can be returned from any one list. For more information, see Where Clauses.

doSelect

The following parameters apply to the doSelect method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
objectType	String	Identifies the object type (factory) to query.
whereClause	String	(Optional) Identifies the where clause for the query.

Parameter	Type	Description
maxRows	Integer	Indicates the maximum number of rows to return. Specify -1 to return all rows. **Note:** Regardless of the integer specified, Unicenter Service Desk will return a maximum of 250 rows per call.
attributes	String[]	Identifies the attribute list for which to fetch values. Dotted-attributes are permitted. If this field is blank, all value-based attributes are returned. These attributes cannot be defined as LOCAL in the majic definition file. LOCAL attributes are temporal; they have no database storage.

Description

Performs an SQL-like select on a specified object table. Supply one or more attributes you want fetched from the objects that match the supplied where clause.

Returns

A sequence of <UDSObject> elements. The following format applies:

```
<UDSObjectList>
    <UDSObject>
     <Handle>
     <Attributes>
    <AttributeNameA>
    <AttributeValueA0>
    <AttributeValueA1>
    <AttributeNameB>
    <AttributeValueB0>
    …
```

XML Element	Type	Description
<UDSObject>	N/A	Specifies the standard UDSObject element containing the handle and requested attribute values.
<UDSObjectList>	Sequence	Contains a <Handle> element and an <Attributes> sequence. For information about the contents of each <UDSObject> element, see XML Object Returns.

For example, if the method used is the following:

```
String[] myArray = [“last_name”, “first_name”]
doSelect(mySID, “cnt”, “last_name LIKE ‘J%’”, 2, myArray)
```

The return could be the following:

```
<UDSObjectList>
    <UDSObject>
    <Handle>cnt:555A043EDDB36D4F97524F2496B35E75</Handle>
    <Attributes>
    <AttributeName>last_name</AttributeName>
    <AttributeValue>Johnson</AttributeValue>
    <AttributeName>first_name</AttributeName>
    <AttributeValue>Carol</AttributeValue>
    </Attributes>

</UDSObject>

<UDSObject>
    <Handle>cnt:555A043EDDB36D4F97524F2496B35E76</Handle>
    <Attributes>
    <AttributeName>last_name</AttributeName>
    <AttributeValue>Jones</AttributeValue>
    <AttributeName>first_name</AttributeName>
    <AttributeValue>Ron</AttributeValue>
    </Attributes>

</UDSObject>

</UDSObjectList>
```

doQuery

The following parameters apply to the doQuery method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
ObjectType	String	Identifies the object type (factory) to query.
WhereClause	String	(Optional) Identifies the where clause for the query.

Description

Performs an SQL-like select for the specified object type. It also returns a *list handle* that points to a list of the rows returned from the query, where each row represents a Unicenter Service Desk object that matched the supplied where clause. The caller can fetch values for the list rows using getListValues ().

Note: For more information about where clauses, see Where Clauses.

The object list is stored on the Unicenter Service Desk server and consumes system resources. The caller is responsible for freeing the list with freeListHandles().

Lists created with this function are homogenous, meaning the objects are all the same type, and they are static, meaning the list never changes even if a data change to an object excludes it from the initial where clause.

Returns

A list handle that must be freed with freeListHandle().

XML Element	Type	Description
<listHandle>	Integer	Identifies the list handle.
<listLength>	Integer	Identifies the length of the list generated.

getListValues

The following parameters apply to the getListValues method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
ListHandle	Integer	Identifies the list handle obtained with a previous call.
StartIndex	Integer	Identifies the position in the list from which to begin fetching.

Parameter	Type	Description
EndIndex	Integer	Identifies the last list position from which to fetch. Specify -1 to fetch all rows from StartIndex. **Note:** Regardless of the integer specified, Web Services will return a maximum of 250 rows per call.
AttributeNames	String[]	Identifies an array of one or more attribute names for which you want to fetch values.

Description

Returns attribute values for a range of objects in a list. For example:

```
< UDSObjectList >
    <UDSObject>
    <Handle>
    <Attributes>
        <AttributeName0>
        <AttributeName1>
```

Returns

The following:

XML Element	Type	Description
<UDSObjectList>	Sequence	Identifies the outer Element, which contains a sequence of <UDSObject> elements.
<UDSObject>	N/A	Contains a <Handle> element and <Attributes> sequence. For information about the contents of each <UDSObject> element, see XML Object Returns.

freeListHandles

The following parameters apply to the freeListHandles method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
Handles	Integer[]	Identifies an array of list handles to free.

Description

Frees the server-side resources for a list and invalidates the list handles. This method should be called whenever a list reference is no longer needed.

Returns

Nothing.

getRelatedList

The following parameters apply to the getRelatedList method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
ObjectHandle	String	Identifies the object handle.
ListName	String	Identifies a list-type attribute name of the object.

Description

Returns a list handle for list (QREL or BREL) attribute of an object. For example, the request object has a related list named “children”, which is a list of its child requests. The Request’s Activity Log (“act_log” or “act_log_all”) is another example.

To retrieve information about an object’s list attributes, refer to the object schema (majic) documentation or use getObjectTypeInformation().

Returns

The following:

XML Element	Type	Description
<listHandle>	Integer	Identifies the list handle.
<listLength>	Integer	Identifies the length of the list generated.

getRelatedListValues

The following parameters apply to the getRelatedListValues method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
objectHandle	String	Identifies the object handle.
listName	String	Identifies a list-type attribute name for the object.
numToFetch	Integer	Signifies the maximum number of rows to return. ▪ Cannot be zero ▪ Specify -1 to return all rows **Note:** Regardless of the integer specified, Web Services will return a maximum of 250 rows per call.
attributes	String[]	Identifies an array of one or more attribute names for which to fetch values. Dotted names are permitted.

Description

Returns values for lists related to a specific object. The lists must be defined as a QREL or BREL. Use the LREL methods to query LREL types.

For example, the request object has a related list named “children”, that is a list of its child requests. This method is a list handle-free alternative to getRelatedList(). The return format is similar to getListValues(), as indicated by the following:

```
<numRowsFound>

< UDSObjectList >
    <UDSObject>
    <Handle>
    <AttributeName0>
    <AttributeName1>
```

To retrieve information for an object’s list attributes (object schema—majic), see Understanding Service Desk Objects and the appendix Data Element Dictionary in the *Unicenter Service Desk Modification Guide*. An alternative method is to use getObjectTypeInformation().

Returns

The following:

XML Element	Type	Description
<numRowsFound>	Integer	Indicates the total number of rows in the queried list. **Note:** This is not necessarily how many rows were returned.
<UDSObjectList>	N/A	Identifies the outer element, which contains a sequence of <UDSObject> elements.
<UDSObject>	N/a	Contains a <Handle> element and zero or more <*AttributeNameX*> elements. For more information about the contents of each <UDSObject> element, see XML Object Returns.

getPendingChangeTaskListForContact

The following parameters apply to the getPendingChangeTaskListForContact method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
contactHandle	String	Identifies the contact handle.

Description

Returns a list handle representing all the “pending” change order workflow tasks assigned to a contact. A “pending” task is an active workflow task with a status that permits task updates.

Returns

The following:

XML Element	Type	Description
<listHandle>	Integer	Identifies the list handle.
<listLength>	Integer	Identifies the length of the list generated.

getPendingIssueTaskListForContact

The following parameters apply to the getPendingIssueTaskListForContact method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
contactHandle	String	Identifies the contact handle.

Description

Returns a list handle representing all the “pending” Issue tasks assigned to a contact. A “pending” task is an active task with a status that permits task updates.

Returns

The following:

XML Element	Type	Description
<listHandle>	Integer	Identifies the list handle.
<listLength>	Integer	Identifies the length of list generated.

getNotificationsForContact

The following parameters apply to the getNotificationsForContact method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
contactHandle	String	Identifies the contact handle.
queryStatus	Integer	(Optional) Identifies the target of the notifications.

Description

Returns a list of notifications (‘lr’ objects) for a contact.

You can query on a specific status for the notifications with the *queryStatus* field, which is useful for returning, for example, only non-cleared messages. The possible *queryStatus* values are as follows:

- -1=Fetch all notifications
- 0=Fetch non-cleared notifications (those with a status value of less than 7)
- 1=Fetch cleared notifications (those with a status value of 7, 8 or 9)

Returns

The following:

XML Element	Type	Description
<listHandle>	Integer	Identifies the list handle.
<listLength>	Integer	Identifies the length of the list generated.

LREL Methods

LREL relationships are defined in majic (.maj) and mod (.mod) files on the server and are declared with the ‘LREL’ keyword. LRELs define many-to-many relationships. The most commonly used LRELs are, for example, relationships between the following:

- Issues and Assets
- Contacts and Assets
- Task Types and Status codes.

Attachments also use LRELs, although the Web Services does not define any Attachment-related methods.

Group relationships do not use LRELs, special methods are provided to group management.

Remember, an LREL has two names, one for each side of the many-to-many relationship. For example, the following is declared in lrels.maj:

```
LREL lrel1 iss asset <> "nr:PDM" issnr ;
```

This declares a many-to-many relationship between the Issue (iss) and Asset (nr) tables. An issue sees the collection of related assets as a pseudo-list called “asset”, while an asset is related to issues as “issnr”. To discover how many assets are related to a particular issue, call getLrelLength with the following parameters:

```
getLrelLength(sid, IssueHandle, “asset”)
```

where *sid* is a SID obtained with login and *IssueHandle* is the string handle to a particular issue object.

Similarly, the following pseudo-code describes how to get the names of all the assets related to an issue:

```
String attrs[] = {“name”};

getLrelValues(sid, IssueHandle, “asset”, 0, -1, attrs);
```

getLrelLength

The following parameters apply to the getLrelLength method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
contextObject	String	Identifies the object on one side of the LREL.
LrelName	String	Identifies the LrelName. Use the name.

Description

Returns the number of objects on one side of a many-to-many relationship:

contextObject

Specifies it as a handle to an object on one side of the LREL relationship.

LrelName

Specifies it as the name of the side of the relationship identified by *ObjHandle*.

Returns

The following:

XML Element	Type	Description
<Length>	Integer	Specifies the number of objects.

getLrelValues

The following parameters apply to the getLrelValues method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
contextObject	String	Identifies the object on one side of the LREL.
LrelName	String	Identifies the Lrel Name. Use the name.
startIndex	Integer	Identifies the position in the “list” from which to begin fetching.
endIndex	Integer	Identifies the Last “list” position from which to fetch. Specify -1 to fetch all rows from startIndex.
attributes	String[]	Identifies an array of one or more attribute names for which to fetch values.

Description

Returns attribute values for a range of objects in an LREL relationship. Remember that items involved in an LREL relationship have no specific ordering. In fact, it is not really a “list”, as defined in this document.

The start and end index parameters are there to help throttle a large number of items. The format is as follows:

```
< UDSObjectList >
    <UDSObject>

    <Handle>

    <AttributeName0>

    <AttributeName1>
```

Returns

The following:

XML Element	Type	Description
<UDSObjectList>	Array	Specifies the outer element, which contains a sequence of <UDSObject> elements.
<UDSObject>	Sequence	Contains a <Handle> element and zero or more *<AttributeNameX>* elements. **Note:** For information about the contents of each <UDSObject> element, see XML Object Returns.

createLrelRelationships

The following parameters apply to the createLrelRelationships method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
contextObject	String	Identifies the object on one side of the LREL.

Parameter	Type	Description
LrelName	String	Identifies the Lrel Name as seen by *contextObject*.
addObectHandles	String[]	Identifies the handles of objects for the other side of the LREL relationships.

Description

Adds one or more many-to-many relationships. *contextObject* is one side of the LREL relation. The caller passes one or more object handles for the other side.

If a relationship already exists between the two objects, no change is made and the system continues to process the *addObjectHandles* array. If an invalid object handle is passed, the entire operation is canceled.

The following example shows how to add several assets to a contact's environment:

```
createLrelRelationships(sid, ContactHandle, “cenv”,

 [“nr:655A043EDDB36D4F97524F2496B35E75”, “nr:755A043EDDB36D4F97524F2496B35E75”])


ContactHandle is a string handle to a contact, and

“nr:655A043EDDB36D4F97524F2496B35E75” and

 “nr:755A043EDDB36D4F97524F2496B35E75” are Asset handles.
```

Returns

Nothing.

removeLrelRelationships

The following parameters apply to the removeLrelRelationships method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
contextObject	String	Identifies the object on one side of the LREL.
LrelName	String	Identifies the Lrel Name (as seen by *contextObject).*
removeObjectHandles	String[]	Identifies the handles of objects to remove from the other side of the LREL relationships.

Description

Removes one or more many-to-many relationships. *contextObject* is one side of the LREL relationship. The caller passes one or more object handles for the other side.

It is not an error if no relationship existed between the two objects. If an invalid object handle is passed, the entire operation is canceled.

For a usage example, see createLrelRelationships() in this chapter.

Returns

Nothing.

Knowledge Tools

To use the Web Services Knowledge Tools, it is helpful if you are familiar with the database structure.

Table Types

Some of the more important tables are described as follows:

Table Type	**Description**
skeleton	Stores all information pertaining to documents with each row representing one document. Field names from this table can be used when passing the PropertyList and SortBy parameters to methods such as FAQ() and Search(). The field names are case-sensitive so make sure you pass them just as they are in the database.
o_indexes	Stores all information pertaining to categories with each row representing one category.

Knowledge Tools General Methods

This section describes Knowledge Tools general methods. Valid Knowledge document sorting properties (when available) are as follows:

- RELEVANCE
- AUTHOR_ID
- BU_RESULT
- CREATION_DATE
- DOC_TYPE_ID
- EXPIRATION_DATE
- HITS
- id
- MODIFY_DATE
- OWNER_ID
- PRIORITY_ID
- ACCEPTED_HITS
- ASSET_ID
- SD_ASSET_ID
- ASSIGNEE_ID
- PRODUCT_ID
- START_DATE
- STATUS_ID
- SUBJECT_EXPERT_ID

faq

The following parameters apply to the faq method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
categoryID	String	Identifies the category ID used to perform the faq. Use 1 for the 'Root' category. **Note:** Multiple ids are supported, for example, "1, 2, 3".

Parameter	**Type**	**Description**
resultSize	Integer	Identifies the number of documents for which you want to retrieve detailed information. For the rest of the documents, only their IDs return. Detailed information for these documents can be accessed later using the getDocumentsByIDs() method. The default is 10.
propertyList	String	Identifies the comma-separated list of database fields from which you want to retrieve information. The following fields are always returned, regardless of the propertyList parameter: ▪ id ▪ DOC_TYPE_ID ▪ BU_RESULT
sortBy	String	Identifies the database field that you want to use for sorting the results. Multiple sort fields are not supported. The default is BU_RESULT, meaning that the faq rating sorts it. When id is used as a secondary sort, it always sorts the results.
descending	Boolean	Identifies an indicator available for sorting the results in descending order.
whereClause	String	Use this to add your own ‘SQL where clause’ for filtering the results of the search.
maxDocIds	Integer	Identifies the maximum amount of document IDs to be returned (the default is 100). For example, if you specify a resultSize of 10 and a maxDocIds of 50, if there are 100 matching documents in the database, then 10 have their detailed information retrieved and 40 have just their IDs returned. The remaining 50 are not returned at all.

Description

Use to perform a faq search. Documents are retrieved based on the category ID that is passed. Any documents residing in that category or in any sub-category are returned. To improve performance, these methods only retrieve detailed information on a user-defined set of documents, which is controlled through the resultSize parameter. The rest of the documents return their IDs only. Using this method, you can for example, set up a paging mechanism where the user can click on 'Top', 'Previous', 'Next', and 'Bottom' links. When you need to retrieve the next set of information, you can use the getDocumentsByIDs() method. The maximum number of 100 IDs is returned.

Returns

A <UDSObjectList> node with the following sections:

<UDSObject> node from the <UDSObject> Node Description section of this chapter. There is a <UDSObject> node with all the given properties for the first *n* documents that the method finds where *n* equals the resultSize parameter.

For example, if the resultSize parameter is 10, the maxDocIDs parameter is 50, and the method finds 100 documents, then there are 10 <UDSObject> with <Attributes> nodes in the first<UDSObject> section with detail attribute information from propertyList parameter and 40 <UDSObject> nodes with only ID <Attributes> in the following section. If you want to retrieve detailed ID <AttrName> information for documents numbering 11-20, you have to make a call to the getDocumentsByIDs() method and pass it those IDs from <AttrValue>.

search

The following parameters apply to the search method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
problem	String	Identifies the problem description to which you want to find solutions.

Parameter	**Type**	**Description**
resultSize	Integer	Identifies the number of documents for which you want to retrieve detailed information. The remaining documents have their IDs returned only. Detailed information for these documents can be accessed later using the getDocumentsByIDs() method. The default is 10.
propertyList	String	Identifies the comma-separated list of database fields from which you want to retrieve information. The following fields are always returned, regardless of the propertyList parameter: ▪ id ▪ DOC_TYPE_ID
sortBy	String	Identifies the database field that you want to use for sorting the results. Multiple sort fields are not supported. The default is RELEVANCE. When id is a secondary sort, it always sorts the results. For a valid sort property, see the faq method.
descending	Boolean	Identifies an indicator you can use for sorting the results in descending order.
getRelatedCategories	Boolean	Returns a list of all related categories for the documents found.
searchType	Integer	Type of search to perform: ▪ 1 = Natural Language Search (NLS) ▪ 2 = Keyword search
matchType	Integer	Represents the type of match: ▪ 0 = OR type match ▪ 1 = AND type match ▪ 2 = Exact match **Note:** If NLS is selected for the searchType parameter, then only the OR and AND matchTypes are valid.

Parameter	Type	Description
searchFields	Integer	Represents the binary combination of fields in which to search: ■ Title = 1 ■ Summary = 2 ■ Problem = 4 ■ Resolution = 8 For example, to search all fields, specify 15 (1+2+4+8). To search in Summary and Problem only, specify 6 (2+4). **Note:** The default is to search Problem. If you set the searchType parameter to NLS, the searchFields parameter is ignored because NLS searches can only search the Problem field.
categoryPaths	String	Limits the results of the search to a specific category or categories. You need to specify the full ID path to the category and separate multiple categories with commas. For example, 1-3-5, 1-4-8 to limit the search to categories 5 and 8 (and their sub-categories).
whereClause	String	Use this to add your own ‘SQL where clause’ for filtering the results of the search.
maxDocIds	Integer	Represents the maximum amount of document IDs allowed to be returned. For example, if you specify a resultSize of 10 and a maxDocIds of 50, if there are 100 matching documents in the database, then 10 have their detailed information retrieved, and 40 have just their IDs returned. The remaining 50 are not returned at all. The default is 100.

Description

Searches for solutions to a problem. Documents are retrieved based on the problem that is passed. Any documents matching the description of the problem or a similar description, are returned. To improve performance, these methods only retrieve detailed information on a user-defined set of documents, which is controlled through the resultSize parameter. The rest of the documents return their ids only. Using this method, you can for example, set up a paging mechanism, where the user can click on ‘Top’, ‘Previous’, ‘Next’, and ‘Bottom’ links. When you need to retrieve the next set of information, you can use the getDocumentsByIDs() method.

Returns

A <UDSObjectList> node with the following sections:

<UDSObject> node from the <UDSObject> Node Description section of this chapter. There will be a <UDSObject> node with all the given properties for the first *n* documents that the method finds, where *n* equals the resultSize parameter. For example, if the resultSize parameter is 10, the maxDocIds parameter is 50, and the method finds 100 documents, then there are 10 <UDSObject> nodes with all the properties requested in the <Attributes> section and 40 <UDSObject> with only the ID property in the <Attributes> section. If you want to retrieve detailed <UDSObject> information for documents 11-20, you need to make a call to the getDocumentsByIDs() method and pass it those IDs.

If the getRelatedCategories parameter is set to True, the <UDSObjectList> node is included in the <Attributes> section for related categories. Each <INDEX_DOC_LINKS> node contains the relational ID of the category, as shown by the following example:

```
1-70
```

doSelectKD

The following parameters apply to the doSelectKD method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
whereClause	String	(Optional) Identifies the where clause for the query.

Parameter	Type	Description
sortBy	String	Identifies the database field that you want to use for sorting the results. Multiple sort fields are not supported. The default is BU_RESULT, meaning that the faq rating sorts it. When id is used as the secondary sort, it always sorts the results.
descending	Boolean	Identifies the indicator available for sorting the results in descending order. Use True for descending and False for ascending the document order.
maxRows	Integer	Indicates the maximum number of rows to return. Specify -1 to return all rows. **Note:** Regardless of the integer specified, Unicenter Service Desk will return a maximum of 250 rows per call.
attributes	String[]	Identifies the attribute list from which to fetch values. Dotted-attributes are permitted. If this field is blank, all value-based attributes are returned. These attributes cannot be defined as LOCAL in the majic definition file. LOCAL attributes are temporal; they have no database storage.
skip	Integer	Identifies the number of knowledge documents to skip from the beginning. Enter zero (0) to return all documents.

Description

Performs an SQL-like select on a Knowledge Document table. Supply one or more attributes you want fetched from the objects that match the supplied where clause.

Returns

A sequence of <UDSObject> elements. The following format applies:

```
<UDSObjectList>
<UDSObject>
 <Handle>
 <Attributes>
<AttributeName0>
<AttributeName1>
```

XML Element	Type	Description
<UDSObject>	Sequence	Contains a <Handle> element and an <Attributes> sequence. For information about the contents of each <UDSObject> element, see XML Object Returns.
<UDSObjectList>		Signifies the outer element, which contains a sequence of <UDSObject> elements.

getDecisionTrees

The following parameters apply to the getDecisionTrees method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
propertyList	String	Identifies the comma-separated list of database fields for which you want to retrieve information. The following fields are always returned, regardless of the propertyList parameter: ■ id ■ DOC_TYPE_ID ■ BU_RESULT
sortBy	String	Identifies the database field that you want to use for sorting the results. The default is id. Multiple sort fields are not supported. If you specify another field, id as a secondary sort always sorts the results.
descending	Boolean	Identifies an indicator available for sorting the results in descending order.

Description

Retrieves all Decision Trees.

Returns

A <UDSObjectList> node with the following sections:

```
<UDSObject> nodes with requested <Attributes> nodes
```

getDocumentsByIDs

The following parameters apply to the getDocumentsByIDs method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
docIds	String	Identifies the comma-separated list of document IDs to retrieve.
propertyList	String	Identifies the comma-separated list of database fields for which you want to retrieve information. The following fields are always returned, regardless of the propertyList parameter: ▪ id ▪ DOC_TYPE_ID ▪ BU_RESULT
sort By	String	Identifies the database field that you want to use for sorting the results. The default is id, but multiple sort fields are not supported. If you specify another field, id as a secondary sort always sorts the results.
descending	Boolean	Identifies an indicator available for sorting the results in descending order.

Description

Retrieves information on one or more documents by passing the document IDs for which you want to retrieve information. This is usually used after calling the faq() or search() methods. In order to improve performance, these methods only retrieve detailed information on a user-defined set of documents. The rest of the documents return their IDs only. For example, you can set up a paging mechanism, where the user can click on Top, Previous, Next, and Bottom links. When you need to retrieve the next set of information, you can use the getDocumentsByIDs() method.

Returns

A <UDSObjectList> node with the following section:

`<UDSObject> nodes with requested <Attributes> nodes describing Knowledge Document`

You should pass the IDs into the getDocumentsByIDs() docIds parameter in this same format.

getDocument

The following parameters apply to the getDocument method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
docId	Integer	Identifies the document ID to retrieve.
propertyList	String	Identifies the comma-separated list of database fields from which you want to retrieve information. Leave blank to retrieve all fields.
getRelatedDocuments	Boolean	Indicates whether to retrieve a list of documents that are related to this document.
getAttachments	Boolean	Indicates whether to retrieve the list of attachments and URL links for the document.
getHistory	Boolean	Indicates whether you want to retrieve the complete history for the document.

Parameter	Type	Description
getComments	Boolean	Indicates whether you want to retrieve all comments for the document.
getNotificationList	Boolean	Indicates whether you want to retrieve the email notification list for the document.

Description

Retrieves information for a document.

Returns

A <UDSObject> node, as described in the <UDSObject> Node Description, with requested <Attributes> provided by the propertList parameter.

createDocument

The following parameters apply to the createDocument method:

Parameter	Type	Description
SID	String	Identifies the session retrieved from logging in.
attributeValues	String[]	Identifies an array of name-value pairs used to set the initial attribute values for the new Knowledge document, as illustrated by the following: `“SUMMARY”,”Summary text”,` "TITLE","Title text"

As part of the createDocument method, the following table reflects examples of valid, commonly used attribute values for a document. Data in the Type column reflect the actual type, which must be parsed to the method in string format in the attrVals string array.

Attribute Value	**Type**	**Description**
PRIMARY_INDEX	Integer	Identifies the category ID in which to create the document. Use 1 for the *Root* category.
USER_DEF_ID	String	Identifies any ID that you would like to use to represent the document.
TITLE	String	Identifies the title of the document.
SUMMARY	String	Identifies the summary of the document.
PROBLEM	String	Identifies the problem of the document.
RESOLUTION	String	Identifies the resolution of the document. This can contain html.
STATUS_ID	Integer	Identifies the status ID for the document. The default is 70 (Published).
PRIORITY_ID	Integer	Identifies the priority ID for the document. The default is 20 (Normal).
CREATION_DATE	Date (String)	Identifies the date and time the document was created. Leave blank to assign current date.
MODIFY_DATE	Date (String)	Identifies the date and time the document was last modified. Leave blank to assign the current date.
START_DATE	Date (String)	Identifies the date the document becomes active and is used in conjunction with Expiration_Date. Leave blank to specify no start date and the document will be active as long as the expiration date has not been reached.

Attribute Value	Type	Description
EXPIRATION_DATE	Date (String)	Identifies the date the document expires, and it is used in conjunction with Start_Date. Leave blank to specify no expiration date.
PUBLISHED_DATE	Date (String)	Identifies the date and time the document was published. Leave blank to assign current date if the status is Published. If the status is not Published, this parameter is ignored.
SD_PRODUCT_ID	Integer	Identifies the product ID from Unicenter Service Desk with which to associate this document.
ASSIGNEE_ID	UUID	Identifies the unique assignee ID from Unicenter Service Desk to which this document is assigned.
SD_ASSET_ID	UUID	Identifies the asset ID from Unicenter Service Desk with which to associate this document.
SD_ROOTCAUSE_ID	Integer	Identifies the root cause ID from Unicenter Service Desk with which to associate this document.
SD_PRIORITY_ID	Integer	Identifies the priority ID from Unicenter Service Desk with which to associate this document.
SD_SEVERITY_ID	Integer	Identifies the severity ID from Unicenter Service Desk with which to associate this document.
SD_IMPACT_ID	Integer	Identifies the impact ID from Unicenter Service Desk with which to associate this document.
SD_URGENCY_ID	Integer	Identifies the urgency ID from Unicenter Service Desk with which to associate this document.
AUTHOR_ID	UUID	Identifies the unique ID of the contact who authored this document. If the author is not an internal contact, you can set this field to zero and use the Author parameter instead.

Attribute Value	Type	Description
OWNER_ID	UUID	Identifies the unique ID of the contact who owns this document.
SUBJECT_EXPERT_ID	UUID	Identifies the unique ID of the contact who is the subject expert for this document.
NOTES	String	Identifies the notes for the document.
READ_GROUP_LIST	String	Identifies the dash-separated list of group IDs that have read permission for this document (for example: 1-3-4). Use *A* to assign permission to everyone.
WRITE_GROUP_LIST	String	Identifies the dash-separated list of group IDs that have write permission for this document (for example: 1-3-4). Use *A* to assign permission to everyone.
INHERITPERMISSIONS	Boolean	Indicates the status of the flag to inherit permissions from the category in which the document is being created. Set to True if you want to inherit permissions, and then ReadPermissions and the WritePermissions parameters will be ignored.
DOC_TYPE_ID	Integer	Identifies the ID for the type of document that this document will be; a regular document or a tree document. The default is a regular document.
HITS	Integer	Identifies the number of times that the document has been viewed.
DOC_TEMPLATE_ID	Integer	Identifies the ID for the template you want to assign to this document.
WF_TEMPLATE	Integer	Identifies the ID for the workflow template you want to assign to this document.
CUSTOM1	String	Specifies a custom field.

Attribute Value	Type	Description
CUSTOM2	String	Specifies a custom field.
CUSTOM3	String	Specifies a custom field.
CUSTOM4	String	Specifies a custom field.
CUSTOM5	String	Specifies a custom field.
CUSTOM_NUM1	Double	Specifies a numeric custom field.
CUSTOM_NUM2	Double	Specifies a numeric custom field.

Description

Creates a new document.

Returns

A <UDSObject> node describing the Knowledge Document created.

modifyDocument

The following parameters apply to the modifyDocument method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
docId	Integer	Identifies the unique ID of the document you want to modify.
attributeValues	String[]	Specifies the name-value pairs for the update, for example, “SUMMARY”, ”Summary text”, “TITLE”, and “Title text”.

As part of the modifyDocument method, the following table reflects examples of valid, commonly used attribute values for a document. Data in the Type column reflect the actual type, which must be parsed to the method in string format in the attrVals string array.

Parameter	Type	Description
MODIFY_DATE	(String)	Indicates a special field used for "record locking" purposes to make sure someone else is not updating the document at the same time that you are. You must pass in the existing MODIFY_DATE of the document. If you leave this blank, you receive an error that another user has updated the document.
USER_DEF_ID	String	Specifies any ID that you want to use to represent the document.
TITLE	String	Indicates the title of the document.
SUMMARY	String	Indicates the summary of the document.
PROBLEM	String	Indicates the problem of the document.
RESOLUTION	String	Indicates the resolution of the document. This can contain html.
STATUS_ID	Integer	Indicates the status ID for the document. The default is 70 (Published).
PRIORTY_ID	Integer	Indicates the priority ID for the document. The default is 20 (Normal).
START_DATE	Date (String)	Indicates the date that the document becomes active, which is also used in conjunction with ExpirationDate. Leave blank to specify no start date and the document becomes active as long as the expiration date is not exceeded.
EXPIRATION_DATE	Date (String)	Indicates the date that the document expires, which is used in conjunction with StartDate. Leave blank to specify no expiration date.

Parameter	Type	Description
SD_PRODUCT_ID	Integer	Indicates the product ID from Unicenter Service Desk with which to associate this document.
ASSIGNEE_ID	UUID	Indicates the unique ID from Unicenter Service Desk to which this document is assigned.
SD_ASSET_ID	UUID	Indicates the asset ID from Unicenter Service Desk with which to associate this document.
SD_ROOTCAUSE_ID	Integer	Indicates the root cause ID from Unicenter Service Desk with which to associate this document.
SD_PRIORITY_ID	Integer	Indicates the priority ID from Unicenter Service Desk with which to associate this document.
SD_SEVERITY_ID	Integer	Indicates the severity ID from Unicenter Service Desk with which to associate this document.
SD_IMPACT_ID	Integer	Indicates the impact ID from Unicenter Service Desk with which to associate this document.
SD_URGENCY_ID	Integer	Specifies the urgency ID from Unicenter Service Desk with which to associate this document.
AUTHOR_ID	UUID	Identifies the unique ID of the contact who authored this document. If the author is not an internal contact, you can set this filed to zero (0) and use the Author parameter instead.
OWNER_ID	UUID	Identifies the unique ID of the contact who owns this document.
SUBJECT_EXPERT_ID	UUID	Indicates the unique ID of the contact who is the subject expert for this document.
NOTES	String	Indicates notes for the document.

Parameter	Type	Description
READ_GROUP_LIST	String	Indicates the dash-separated list of group ids that have read permission for this document (for example: 1-3-4). Use *A* to assign permission to everyone.
WRITE_GROUP_LIST	String	Indicates the dash-separated list of group IDs that have write permission for this document (for example: 1-3-4). Use *A* to assign permission to everyone.
INHERIT_PERMISSION	Boolean	Indicates the status of the inherit permissions flag. Set to True if you want to inherit permissions from the category in which the document is being created. If set to True, the ReadPermissions and WritePermissions parameters are ignored.
DOC_TYPE_ID	Integer	Identifies the ID for the type of document that this document is to become; a regular document or a tree document. The default is a regular document.
HITS	Integer	Indicates the number of times that the document has been viewed.
DOC_TEMPLATE_ID	Integer	Identifies the ID for the template you want to assign to this document.
WF_TEMPLATE	Integer	Identifies the ID for the workflow template you want to assign to this document.
CUSTOM1	String	Indicates a custom field.
CUSTOM2	String	Indicates a custom field.
CUSTOM3	String	Indicates a custom field.
CUSTOM4	String	Indicates a custom field.
CUSTOM5	String	Indicates a custom field.
CUSTOMNUM1	Double	Indicates a numeric custom field.
CUSTOMNUM2	Double	Indicates a numeric custom field.

Description

Modifies a document.

Returns

A <UDSObject> node describing the Knowledge Document modified.

deleteDocument

The following parameters apply to the deleteDocument method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
docId	Integer	Identifies the unique ID of the document you want to delete.

Description

Flags a document for deletion. The Knowledge Tools Windows Service permanently deletes the document.

Returns

Returns error codes only when there are *individual* errors. For additional information, see Error Codes.

getCategory

The following parameters apply to the getCategory method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
categoryId	Integer	Identifies the category ID in which to create the document.
getCategoryPaths	Boolean	Indicates the path for which to get category information. It returns category information and the full text category path for each category.

Description

Retrieves information for a category, including a listing of all of its child categories.

Returns

A <UDSObjectList> node with an <UDSObject> node describing the category requested.

getComments

The following parameters apply to the getComments method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
docIds	String	Identifies the document IDs for all the comments you want to retrieve. **Note:** Use commas to separate, for example, "400001,400002".

Description

Gets all the comments from documents.

Returns

A <UDSObjectList> node with zero or more <UDSObject> nodes describing O_COMMENTS with the following <Attributes> child nodes:

XML Element Value	Data Type	Description
id	Integer	Identifies the unique identifier for the comment most recently added.
DOC_ID	Integer	Identifies the document IDs for the comment recently added.
USER_ID	Integer	Identifies the ID of the person who submitted the comment.
USER_NAME	String	Identifies the user name of the person who submitted the comment.
EMAIL_ADDRESS	String	Identifies the email address of the person who submitted the comment.
COMMENT_TEXT	String	Identifies the text for the comment recently added.
COMMENT_TIMESTAMP	Date	Identifies the date and time the comment was added.

addComment

The following parameters apply to the addComment method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
comment	String	Identifies the comment to add.

Parameter	Type	Description
docId	Integer	Identifies the document ID for the comment you want to add.
email	String	Indicates the email address of the person who submitted the comment. Leave blank if you want the email address retrieved from the database based on the user ID parameter.
username	String	Indicates the user name of the person who submitted the comment. Leave blank if you want the user name retrieved from the database based on the user id parameter.
contactId	String	Indicates the ID of the person submitting the comment. If this contact ID exists in the database, the associated email and user name are retrieved and placed in the email and user name fields. If this ID does not exist, the email and user name parameters are used instead, if supplied. Use zero (0) if you are not using this parameter. Contact ID is UUID in string format.

Description

Adds a comment to a particular document.

Returns

A <UDSObject> node with the following<Attributes> child nodes describing the comment most recently added:

XML Element Value	Data Type	Description
id	Integer	Identifies the unique identifier for the comment most recently added.
DOC_ID	Integer	Identifies the document ID for the comment most recently added.
USER_ID	Integer	Identifies the ID of the person who submitted the comment.
USER_NAME	String	Identifies the user name of the person who submitted the comment.

XML Element Value	Data Type	Description
EMAIL_ADDRESS	String	Identifies the email address of the person who submitted the comment.
COMMENT_TEXT	String	Identifies the text for the comment recently added.
COMMENT_TIMESTAMP	Date	Identifies the date and time the comment was added.

deleteComment

The following parameters apply to the deleteComment method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
commentId	Integer	Identifies the unique ID for the comment you want to delete.

Description

Deletes a comment.

Returns

Returns error codes only for *individual* errors. For additional information, see Error Codes.

rateDocument

The following parameters apply to the rateDocument method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
docId	Integer	Identifies the document ID to rate.
rating	Integer	Identifies the rating to give the document (a scale of 0-4, where 0 is the worst and 4 is the best).
ticketPersId	String	Identifies the persistent ID of a ticket related to this Knowledge document.

Parameter	Type	Description
multiplier	Integer	Identifies the multiplier parameter. This parameter can be used to simulate many ratings at once. Use the default of 1 for a single rating and any other number for multiples. For example, if you submit 3, it acts as if you called the method 3 times. Three ratings with the value you supplied in the rating parameter is added to the database.
onTicketAccept	Boolean	Identifies whether the document was accepted as a solution for the ticket.
solveUserProblem	Boolean	Identifies whether this document solved the user's problem. It signifies how the user responded to the question "Did this document solve your problem?" on the Solution Survey.
isDefault	Boolean	Indicates a default rating status. If you are setting the rating just because the user viewed the document and not because he actually rated it, set this to True. This is used for reporting reasons.

Description

Rates a particular document.

Returns

A <UDSObject> node with the following <Attributes> children nodes describing the rating BU_TRANS :

XML Element Value	Data Type	Description
id	Integer	Identifies the unique identifier for the rating most recently added. Use this with the updateRating() method if you want to modify the rating at a later time.
DOC_ID	Integer	Identifies the document ID.
INDEX_ID	Integer	Identifies the category ID.

XML Element Value	Data Type	Description
BU_RATING	String	Identifies the rating given to the document.
HIT_NO_VOTE	Integer	Identifies the rating set because a user viewed the document and not actually rated it, or vice versa.

updateRating

The following parameters apply to the updateRating method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
ratingID	Integer	Identifies the unique ID of the rating you want to modify. This ID is returned by the rateDocument() method.
rating	Integer	Identifies the new rating to apply to the document (a scale of 0—4, where 0 is the worst and 4 is the best).

Description

Updates one of the ratings of a particular document.

Returns

A <UDSObject> node describing BU_TRANS with the updated rating attribute.

getQuestionsAsked

The following parameters apply to the getQuestionsAsked method:

Parameter	Type	Description
SID	String	Identifies the session retrieved from logging in.
resultSize	Integer	Identifies the number of searched text for which you want to retrieve detailed information.

Parameter	Type	Description
Descending	Boolean	Indicates an option available for sorting the results in descending order.

Description

Retrieves historical Knowledge document search text.

Returns

A <UDSObjectList> node with zero or more <UDSObject> nodes describing EBR_LOG with the following <Attributes> child nodes:

XML Element Value	Data Type	Description
id	Integer	Indicates the unique identifier of the question asked.
SEARCH_TEXT	Integer	Indicates the search text of the question asked.

getBookmarks

The following parameters apply to the getBookmarks method:

Parameter	Type	Description
SID	String	Identifies the session retrieved from logging in.
contactId	String	Identifies the unique ID of the contact for which you want to retrieve bookmarks. Contact ID is UUID in string format.

Description

Retrieves bookmarks for a particular contact.

Returns

A <UDSObjectList> node with zero or more <UDSObject> nodes describing CI_BOOKMARKS with the following <Attributes> child nodes:

XML Element Name	Type	Description
DOCUMENT_ID	Integer	Identifies the unique ID of the document.
id	Integer	Identifies the bookmark ID
USER_ID	String	Identifies the user ID for the owner of this bookmark.
BOOKMARK_TITLE	String	Identifies the bookmark title from the document.

addBookmark

The following parameters apply to the addBookmark method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
contactId	String	Identifies the unique ID of the contact for which you want to retrieve bookmarks. Contact ID is UUID in string format.
docId	Integer	Identifies the document ID you want to bookmark.

Description

Adds a bookmark for a particular contact.

Returns

A <UDSObject> node describing the newly created bookmark.

deleteBookmark

The following parameters apply to the deleteBookmark method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
contactId	String	Identifies the unique ID of the contact for which you want to delete a bookmark. Contact ID is UUID in string format
docId	Integer	Identifies the document ID of the bookmark you want to remove.

Description

Deletes a bookmark for a particular contact.

Returns

Returns error codes only for *individual* errors. For additional information, see Error Codes.

getStatuses

The following parameter applies to the getStatuses method:

Parameter	Type	Description
SID	String	Identifies the session retrieved from logging in.

Description

Retrieves the list of statuses.

Returns

A <UDSObjectList> node with zero or more <UDSObject> nodes describing CI_STATUSES with the following <Attributes> child nodes:

XML Element Values	Data Type	Description
id	Integer	Identifies the unique ID of the status.
STATUS	String	Identifies the name for the status.
STATUS_DESCRIPTION	String	Identifies the description for the status.
PREDEFINED	Integer	Indicates whether the status is predefined by the Knowledge Tools system, meaning that it cannot be deleted.
STATUS_ORDER	Integer	Describes the order by which the status should appear in the Workflow task list. Workflows can only be created when they follow this order.

getPriorities

The following parameter applies to the getPriorities method:

Parameter	Type	Description
SID	String	Identifies the session retrieved from logging in.

Description

Retrieves the list of priorities.

Returns

A <UDSObjectList> node with zero or more <UDSObject> nodes describing CI_PRIORITIES with the following <Attributes> child nodes:

XML Element Values	Data Type	Description
id	Integer	Identifies the unique ID of the priority.

XML Element Values	Data Type	Description
PRIORITY	String	Identifies the name for the priority.

getDocumentTypes

The following parameter applies to the getDocumentTypes method:

Parameter	Type	Description
SID	String	Identifies the session retrieved from logging in.

Description

Retrieves the list of document types.

Returns

A <UDSObjectList> node with zero or more <UDSObject> nodes describing CI_DOC_TYPES with the following <Attributes> child nodes:

XML Element Values	Data Type	Description
id	Integer	Identifies the unique ID of the document type.
DOC_TYPE_TXT	String	Identifies the name for the document type.

getTemplateList

The following parameter applies to the getTemplateList method:

Parameter	Type	Description
SID	String	Identifies the session retrieved from logging in.

Description

Retrieves the list of document templates

Returns

A <UDSObjectList> node with zero or more <UDSObject> nodes describing CI_DOC_TEMPLATES with the following <Attributes> child nodes:

XML Element Value	Type	Description
id	Integer	Identifies the unique ID of the document type.
TEMPLATE_NAME	String	Identifies the name for the document template.
IS_PREDEFINED	Integer	Indicates whether the template is predefined by the Knowledge Tools system and cannot be deleted.
IS_DEFAULT	Integer	Indicates whether the template is the default that will be assigned to new documents.

getWorkflowTemplateList

The following parameter applies to the getWorkflowTemplateList method:

Parameter	Type	Description
SID	String	Identifies the session retrieved from logging in.

Description

Retrieves the list of workflow templates.

Returns

A <UDSObjectList> node with zero or more <UDSObject> nodes describing WF_TEMPLATE with the following <Atttributes> child nodes:

XML Element Value	Type	Description
id	Integer	Identifies the unique ID of the workflow template.

XML Element Value	Type	Description
WF_NAME	String	Identifies the name for the workflow template.
WF_DESCRIPTION	String	Identifies the description for the workflow template.
IS_DEFAULT	Integer	Indicates that the default template is to be assigned to new documents.

Use the Knowledge Tools Web Services

The login process and any error codes that may display for the Knowledge Tools Web Services are the same as those found for the Service Desk Web Services. For additional information, see Login and Error Codes.

Access the Knowledge Tools Web Services

The Knowledge Tools Web Services uses Apache Axis implementation of standards set forth by the W3C. Ideally, a client on any type of platform should be able to access the services, but vendor implementations vary. For example, Java and .NET both provide tools for generating proxy classes from a WSDL service description. If you experience any issues using the Web Services with a different technology, consult your platform vendor or Microsoft's knowledge base.

Attachment Methods

This section describes the Web Services Knowledge attachment methods.

createAttmnt

The following parameters apply to the createAttmnt method:

Parameter	Data Type	Description
SID	Integer	Identifies the session retrieved from logging in.
repositoryHandle	String	Identifies the object handle of a document repository.
folderId	Integer	Identifies the folder handle ID.
objectHandle	String	Identifies the object handle of a Knowledge document to which this attachment is attached.

Parameter	Data Type	Description
description	String	Identifies the description for the attachment object.
fileName	String	Identifies the name of the full path of the file to be uploaded.

Description

Uploads a file to the back-end server. An uploaded file is stored in a document repository specified by the repositoryHandle. An attachment object is then created and attached to a document object specified by the objectHandle. The attachment object has all the information for accessing the newly uploaded file in the repository.

Returns

The following:

Parameter	Type	Description
<Handle>	String	Identifies the object handle of the newly created attachment object.

attmntFolderLinkCount

The following parameters apply to the attmntFolderLinkCount method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
folderId	Integer	Identifies the unique ID of the folder

Description

Describes the number of attachment links under a specific folder to be attached.

Returns

Returns the number of attachments found under the specific folder.

attachURLLink

The following parameters apply to the attachURLLink method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
docId	Integer	Identifies the unique ID of the Knowledge document.
URL	String	Indicates the URL to attach to the Knowledge document.
attmntName	String	Identifies the name of the attachment.
Description	String	Indicates the description of the attachment.

Description

Attaches a URL link to a Knowledge document.

Returns

Returns error codes only for *individual* errors. For additional information, see Error Codes.

getKDListPerAttmnt

The following parameters apply to the getKDListPerAttmnt method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
attmntId	Integer	Identifies the unique ID of the attachment.

Description

Returns a list of Knowledge documents with reference to a given attachment.

Returns

A <UDSObject> node with zero or more <UDSObject> nodes describing the Knowledge document with the following <Attributes> child nodes:

XML Element Value	Type	Description
id	Integer	Identifies the unique ID of the Knowledge document.

getAttmntListPerKD

The following parameters apply to the getAttmntListPerKD method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
docId	Integer	Identifies the unique ID of the Knowledge document.

Description

A list of attachments with reference to a given Knowledge document.

Returns

A <UDSObject> node with zero or more <UDSObject> nodes describing the attachment with the following <Attributes> child nodes:

XML Element Value	Type	Description
id	Integer	Identifies the unique ID of the attachment

isAttmntLinkedKD

The following parameters apply to the AttmntLinkedKD method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
attmntId	Integer	Identifies the unique ID of the attachment.

Description

Checks if an attachment is linked to any Knowledge document, and returns the number of links found.

Returns

Any number of links found.

createFolder

The following parameters apply to createFolder method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
parentFolderId	String	Identifies the unique ID of the parent folder (zero if no parent).

Parameter	Type	Description
repId	Integer	Identifies the unique ID of the repository.
folderType	Integer	Identifies the type of folder to be created.
description	String	Identifies the description of the folder.
folderName	String	Identifies the name of the folder.

Description

Creates a folder.

Returns

A <UDSObject> describing the folder created, with some of the following child <Attributes> nodes:

XML Element Value	Type	Description
repository	String	Identifies the repository name.
parent_id	String	Identifies the unique ID of the parent folder.
folder_type	Integer	Identifies the type of folder created.
folder_name	String	Identifies the name of the folder.
description	String	Identifies the description of the folder.

getFolderList

The following parameters apply to getFolderList method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
parentFolderId	String	Identifies the unique ID of the parent folder (0 if no parent).
repId	Integer	Identifies the unique ID of the repository.

Description

Returns a list of folders under a given parent folder

Returns

A <UDSObjectList> with zero or more <UDSObject> describing the attachment folder, with some of the following child <Attributes> nodes:

XML Element Value	Type	Description
repository	String	Identifies the repository name.
parent_id	String	Identifies the unique ID of the parent folder.
folder_type	Integer	Identifies the type of folder created.
folder_name	String	Identifies the name of the folder.
description	String	Identifies the description of the folder.

getFolderInfo

The following parameters apply to the getFolderInfo method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
folderId	Integer	Identifies the unique ID of the attachment folder.

Description

Returns the attributes of a folder.

Returns

A <UDSObject> describing the attachment folder, with some of the following child <Attributes> nodes:

XML Element Value	Type	Description
repository	String	Identifies the repository name.
parent_id	String	Identifies the unique ID of the parent attachment folder.

XML Element Value	Type	Description
folder_type	Integer	Identifies the type of attachment folder.
folder_name	String	Identifies the name of the attachment folder.
description	String	Identifies the description of the attachment folder.

getAttmntList

The following parameters apply to the getAttmntList method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
folderId	String	Identifies the unique ID of the attachment folder.
repId	Integer	Identifies the unique ID of the repository. This parameter is required only when the folder ID is zero, which indicates the root folder.

Description

Returns a list of attachments under a given attachment folder.

Returns

A <UDSObjectList> with zero or more <UDSObject> describing the attachment, with some of the following child <Attributes> nodes:

XML Element Value	Type	Description
repository	String	Identifies the repository name.
parent_id	String	Identifies the unique ID of the parent folder.
folder_type	Integer	Identifies the type of folder created.

XML Element Value	Type	Description
folder_name	String	Identifies the name of the folder.
description	String	Identifies the description of the folder.

getAttmntInfo

The following parameters apply to the getAttmntInfo method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
attmntId	Integer	Identifies the unique ID of the attachment folder.

Description

Returns the attributes of an attachment.

Returns

A <UDSObject> describing the attachment, with some of the following child <Attributes> nodes:

XML Element Value	Type	Description
id	Integer	Identifies the unique ID of the attachment.
description	String	Identifies the description of the attachment.
attmnt_name	String	Identifies the name of the attachment.
file_name	String	Identifies the name of the file.
folder_id	Integer	Identifies the unique ID of the attachment folder.
repository	Integer	Identifies the unique ID of the repository.

getRepositoryInfo

The following parameters apply to the getRepositoryInfo method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
repositoryId	Integer	Identifies the unique ID of the repository.

Description

Returns the attributes of a repository.

Returns

A <UDSObject> describing the repository, with some of the following child <Attributes> nodes:

XML Element Value	Type	Description
repository	String	Identifies the repository name.
parent_id	String	Identifies the unique ID of the parent attachment folder.
folder_type	Integer	Identifies the type of attachment folder.
folder_name	String	Identifies the name of the attachment folder.
description	String	Identifies the description of the attachment folder.

Miscellaneous Methods

This section describes the Web Services Miscellaneous methods.

callServerMethod

The following parameters apply to the callServerMethod method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
methodName	String	Identifies the name of the method to call.
factoryName	String	Identifies the factory name of the object type containing the method.
formatList	String	Identifies the format list, a series of characters describing the intended data types for the incoming parameters. It is related to the Parameter() description in this table).
Parameter0...parameterX	String[]	Indicates zero or more parameter values for the method.

Description

Use this method to invoke an arbitrary server-side method. These are methods defined in the proprietary “spell” scripting language.

Only factory methods can be called and the caller must be logged in with full administrative rights.

The format list is a series of zero or more characters that indicate (in order) the data types of the parameters to follow. The character codes are as follows:

- 0 S—string
- 0 I—integer (covers dates and duration)
- 0 N—null

For example, suppose a spell method is defined as follows:

```
cr::DoStuff(int in_one, string in_two, string in_three);
```

Invoke it with the following:

```
callServerMethod(“DoStuff”, “cr”, “ISS”, [3, “a string”, “another one”]);
```

This method is intended for CA Development and services for customizations only; it is not recommended for most sites.

Returns

Each return message component in its own XML element. The elements are all string representations of the value. The elements are ordered in the return order from the server using the following format:

```
<ServerReturn>
    <Paramx>
```

This call does not support object reference returns. If an object reference is returned by the spell method, the return data is the string, “OBJECT”. This is not an error and any other parameters are also returned.

XML Element	Type	Description
<ServerReturn>	N/A	Indicates the outer element, which contains zero or more <ParamX> elements for return values.
<Paramx>	String	Indicates zero or more for the return values, where *x* is an integer starting at zero and increments for each return element.

createObject

The following parameters apply to the createObject method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
ObjectType	String	Identifies the type of object to create (the majic factory name).
attrVals	String[]	Identifies a sequence of name-value pairs used to set the initial attribute values for the new issue. Dotted names are not permitted.
attributes	String[]	Identifies a sequence of attribute names from the new object for which to return values. Dot-notation is permitted. If this field is empty, all attribute values are returned.

Description

Creates a Service Desk object. The caller is responsible for setting any required fields in the *attrVals* parameter. Dotted-names are not permitted.

ObjectType

Identifies the name for an object type (factory).

attrVals

Describes the array of name-value pairs used to initialize the new object. For example, the following pseudo-code shows how to create a new contact and return a <UDSObject> element with values for all its attributes:

```
String [4] attrVals;

attrVals[0] = “first_name”;  // attribute name

attrVals[1] = “Edgar”;

attrVals[2] = “last_name”;

attrVals[3] = “Martin”;

string [0] emptyArray;

int err = CreateObject(sid, “cnt”, attrVals,emptyArray);
```

Note: Do not use this method for new assets, issues, requests or change orders. Use the specialized createXXX() methods for those object types. This also applies if you are using the ITIL methodology—use the appropriate methods to create Configuration Items, Incidents, and Problems.

Returns

A <UDSObject> element containing the new object’s handle, along with attribute values specified in the *attributes* parameter. If the *attributes* parameter is empty, *all* attribute values are returned. List and LREL types are also returned, but as empty elements.

XML Element	Type	Description
<UDSObject>	N/A	Identifies the standard UDSObject element containing the handle and requested attribute values.
<newHandle>	String	Identifies the new object’s handle.

getBopsid

The following parameters apply to the getBopsid method:

Parameter	Type	Description
SID	Integer	Identifies the SID of the current login session.
Name	String	Identifies the name of the user associated with the returned bopsid. **Note:** This is the system login name, not the Service Desk contact name.

Description

Facilitate the building of Web interface URLs which may be used to launch the web interface in the context of a given user without a login challenge. To launch the web interface in the context of a given user (for example, an analyst), a calling application must first construct a Web interface URL, which includes a bopsid token (a web-interface security token). Failure to provide a bopsid token may result in an interactive login challenge when attempting to launch the web interface in the chosen context (such as, a detail view of a given ticket). The getBopsid method allows the bopsid to be generated in the context of the user provided by the Name parameter. If Name is not provided, it uses the user associated with the current web interface SID.

Note: To prevent unauthorized elevation of privileges, the SID of the current login must have equal or greater access rights than the name of the user entered.

Returns

A bopsid based on the name of the user entered. The returned bopsid expires based on web interface configuration settings and should not be held by the caller indefinitely, but used in a launch of the web interface in a timely manner.

Note: The SID of the current login must have equal or greater access rights than the name of the user entered. If no user name is entered, then the bopsid created is based only on the SID.

getConfigurationMode

The following parameters apply to the getConfigurationMode method:

Parameter	Type	Description
SID	Integer	Identifies the SID of the current login session.

Description

This method returns a string indicating if the Service Desk installation is in the ITIL mode.

Returns

A string "itil" if the installation is in ITIL mode. Otherwise, an empty string is returned.

getObjectValues

The following parameters apply to the getObjectValues method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
objectHandle	String	Identifies the handle of a Service Desk object to query.
attributes	String[]	Identifies the names of attributes to fetch. Dotted-names are permitted. If this field is empty, all the object’s attributes are returned.

Description

This method returns the attribute values of an object. The caller passes one or more attribute names to fetch the object and dotted-names are permitted.

All values are returned as a string. Empty/null attributes are returned as empty strings.

Returns

A <UDSObject> element. For more information, see XML Object Returns.

XML Element	Type	Description
<UDSObject>	N/A	Contains a <Handle> element and zero or more <AttributeNameX> elements.

getObjectTypeInformation

The following parameters apply to the getObjectTypeInformation method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
factory	String	Identifies the object type (known as 'factory') to query. This is the majic name of the object, for example: "cr" = Request

Description

A list of all attribute names for a given object type, along with type information for each attribute. Information returned for the attribute's type includes the Integer, String, Date, Pointer, List, and so on, if the attribute is required for back-storing its storage space requirements (if appropriate).

Callers should cache the type information requested per object type to avoid multiple, redundant (and expensive) calls. The attribute information can change only after modifications are performed on the Service Desk server and the service is recycled.

Returns

The following:

XML Element	Type	Description
<UDSObject>	N/A	Indicates the root node.
<Attributes>	Sequence	Indicates zero or more elements for each attribute.
<attrName DataType=”dataType” Size=”storageSize” Required=”Boolean” Factory=”factoryName”>	Empty Element	Indicates an element with a name matching an object attribute name. The element has several attributes: **DataType** Signifies the integer representation of the data type. For more information about data types, see the table in Attribute Data Types. **Size** Represents the maximum size needed to store this attribute in a string. **Required** Represents the flag status of True if this attribute must be set for the object to back store. **Factory** Represents the Type name of the object if the attribute is a List, Lrel or Pointer type. It is not written unless it is a Llist, Lrel, or Pointer type data type.

serverStatus

The following parameter applies to the serverStatus method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.

Description

Returns the status of the Service Desk server, that is, whether it is up and ready or shut down.

Note: This method executes rapidly on the server machine. Calling this periodically is a good way to keep a SID active.

Returns

The following values apply:

- 1 =Indicates the Service Desk server is not available
- 0= Indicates the Service Desk server it is running

XML Element	Type	Description
<ServerStatus>	Integer	Identifies the value associated with the server status, zero or 1.

updateObject

The following parameters apply to the updateObject method:

Parameter	Type	Description
SID	Integer	Identifies the session retrieved from logging in.
objectHandle	String	Identifies the handle of a Service Desk object to update.

Parameter	Type	Description
attributeValues	String[]	Identifies the name-value pairs for the update.
attributes	String[]	Identifies the sequence of attribute names from the object for which to return values. Dot-notation is permitted. If this field is empty, all attribute values are returned.

Description

Updates one or more attributes for the specified object.

To set values for the object, the caller passes a single-dimensional array of attribute name-value pairs. The first half of the pair is an attribute name; the second is the actual value. Dotted-names are not permitted.

To update an attribute that is a Pointer type (for example, the customer field on a request) a handle must be used for the value. For Integer, Date, and Duration types, pass the string representation of an integer.

For example, to update a request with a new assignee, description and priority, the array would appear as follows:

```
[0] – "assignee"

[1] – "cnt:555A043EDDB36D4F97524F2496B35E75"  (a contact Handle)

[2] – "description"

[3] – "My new description"

[4] – "priority"

[5] – "pri:38903"  (a priority Handle)
```

If the update fails for any reason, the entire operation aborts and no changes occur.

Note: When updating a task, set the status value last in the attribute array.

Returns

A <UDSObject> element containing the updated object's handle, along with attribute values specified in the *attributes* parameter. If the *attributes* parameter is empty, *all* of the attribute values are returned. List and LREL types are also returned, but as empty elements. For more information about the return format, see XML Object Returns.

WorkFlow Methods

This section describes the Web Services workflow-related methods. These methods are used only with Unicenter Service Desk's built-in workflow engine. These methods are not applicable when a ticket is using the integrated CA Workflow product. CA Workflow does have a separate web service interface. For details, see the product documentation.

getWorkflowTemplates

The following parameters apply to the getWorkflowTemplates method:

Parameter	Type	Description
SID	Integer	Identifies the SID of the current login session.
objectHandle	String	Identifies the object handle (persistent_id) of a change order or issue.

Description

Takes a parameter of the handle of a change order or issue and returns a list of workflow templates that are associated with the category of the change order or issue. It first verifies that the change order or issue has a category. Then, it retrieves a list of associated workflow templates of the category. If the change order or issue does not have a category, the following error appears:

```
Error (UDS_BAD_PARAM)
```

Empty/null list is returned if there are no associated workflow templates found for the category

Returns

The following:

XML Element	Type	Description
<listHandle>	String	Identifies the list handle.
listLength	Integer	Identifies the length of the list generated.

createWorkFlowTask

The following parameters apply to the createWorkFlowTask method:

Parameter	**Type**	**Description**
SID	Integer	Identifies the SID of the current login session.
attrVals	String[]	Identifies an array of name-value pairs that are used to set the initial attribute values for this workflow task. These optional attributes include: ▪ Asset ▪ Assignee ▪ Estimated duration ▪ Estimated cost ▪ Estimated completion date
objectHandle	String	Identifies the object handle of a change order or issue for which this workflow task is created or attached.
creatorHandle	String	Identifies the object handle of the contact that created this workflow task.

Parameter	Type	Description
selectedWorkFlow	String	Identifies the object handle of an existing workflow task after which the new workflow task is to be inserted.
taskType	String	Identifies the object handle of a workflow task template or the code of a task type.

Description

Verifies the input value of selectedWorkFlow. If it is empty, the new workflow task is inserted at the bottom. Otherwise, the new workflow task is created after the selectedWorkFlow. The input value of taskType is also verified to determine whether to create a workflow based on a workflow template or on a task type code.

Returns

The new workflow task's handle, along with *all* of its attribute values. List and LREL types are also returned, but as empty elements. For more information about the return format, see XML Object Returns.

XML Element	Type	Description
<UDSObject>	N/A	Identifies the standard UDSObject element containing the handle and requested attribute values.
<NewWorkflowHandle>	String	Identifies the new workflow task's handle.

deleteWorkFlowTask

The following parameters apply to the deleteWorkFlowTask method:

Parameter	Data Type	Description
SID	Integer	Identifies the SID of the current login session.
workflowHandle	String	Identifies the object handle of a workflow task.
objectHandle	String	Identifies the object handle of a change order or issue from which the workflow task is deleted.

Description

Removes a workflow from its associated change order or issue.

Returns

Nothing.

Attachment Methods

This section describes the Web Services attachment-related methods. Only file attachments are handled by these methods; link type attachments are handled by generic methods, such as CreateObject(). In addition, file uploads through Web Services employ the Direct Internet Message Encapsulation (DIME) protocol. Your SOAP implementation must support DIME in order to use these methods.

createAttachment

The following parameters apply to the createAttachment method:

Parameter	Data Type	Description
SID	Integer	Identifies the session retrieved from logging in.
repositoryHandle	String	Identifies the object handle of a document repository.

Parameter	Data Type	Description
objectHandle	String	Identifies the object handle of a call request, change order, or issue, to which this attachment is attached.
description	String	Identifies the description for the attachment object.
fileName	String	Identifies the full path of the file to be uploaded.

Description

Uploads a file to the back-end server. An uploaded file is stored in a document repository specified by the repositoryHandle. An attachment object is then created and attached to a ticket object specified by the objectHandle. The attachment object has all the information for accessing the newly uploaded file in the repository.

Returns

The following:

Parameter	Type	Description
<Handle>	String	Identifies the object handle of the newly created attachment object.

removeAttachment

The following parameters apply to the removeAttachment method:

Parameter	Type	Description
SID	Integer	Identifies the SID of the current login session.
attachmentHandle	String	Identifies the object handle of an attachment to be removed.

Description

Removes an attachment from a ticket object. The attached file is then removed from the repository.

Returns

Nothing.

Index