MICRO FOCUS

Service Desk 7.5

ZENworks Service Desk Developer Resources

March 2018

Legal Notices

Contents

About This Guide

This document includes ZENworks Service Desk developer resources.

Audience

This guide is intended for developers.

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature at the bottom of each page of the online documentation.

Additional Documentation

ZENworks Service Desk is supported by other documentation that you can use to learn about and implement the product. For additional documentation, see the ZENworks Service Desk documentation site (https://www.novell.com/documentation/zenworks-service-desk-75/).

1 ZENworks Service Desk Developer Resources

This appendix contains some useful ZENworks Service Desk developer resources.

1.1 Developer

1.1.1 Integrations

Service Desk provides a rich experience for developers to extend the functionality and integrate with other 3rd party systems. We are continually adding new examples and modules that users can download directly as well as within the product itself.

The SOAP API reference shows how to communicate directly with service desk.There are many different ways to extend service desk depending upon your needs. The most obvious way is through Web Services, but there are many other layers that maybe more appropriate and involve no coding at all.

1.1.2 Connectors

service desk AMIE engine (Asset Management Integration Engine) provides a unique mechanism to connect to any third party Asset Management system.By default service desk provides many built-in connectors. This is expanding every release. One of the unique properties of connectors built using AMIE is that they readily adapt to schema changes on the foreign host so in most cases there is no need to change them between upgrades.

Users have the ability to create their own mapping files and we provide some great examples on how to do this step by step in AMIE Part I and AMIE Part II.

1.2 About API

1.2.1 Version

Syntax

```
version()
```

Description

This API returns the current version of Service Desk that is running.

Parameters

None

Return Value

The following values are returned by this API:

1. Version: Contains the full version string.
2. Release: Contains the release number.
3. Vendor: Contains the OEM vendor name of the solution.
4. serverTzName: Is the server time zone name.
5. serverTzOffset: Is the GMT offset of the time zone.

1.3 Authenticate API

1.3.1 Connect

Syntax

connect (String username, String password)

connect (String sourceId, String username, String password)

Description

This API logs you into the system. You must be logged in before you can call most other functions.

All web services calls, including Connect, observe the session time out constraints defined at the container and application levels. In order to maintain the session flags, the ‘maintainSession’ flag must be set to true on the org.apache.axis.client.Service object using the service. If managing session state manually, the http headers must be preserved for all subsequent interactions. Failure to maintain the headers will result in errors stating that the current session is not logged in.

Parameters

The following are the input parameters for this API:

1. username: A unique username for this account.
2. password: A unique password for this account.
3. sourceId: The sourceId of the LDAP or AD source. Use getLdapSources() to return a list of all known authentication servers. If using PHP or a language that does not use method overloading use “-1″ as the sourceId to simulate null.

Return Value

The following functions should be called to get return values for this API:

1. To get the value of the ‘success’ field, use `get("success")`. The value returned in the ‘success’ field will be a string representation of a boolean (‘true or false’) indicating whether or not the login attempt was successful. If the login attempt fails for any reason the ‘success’ field is ‘false’.
2. To get the value of the ‘message’ field, use `get("message")`. The value returned in the ‘message’ field will be a text message that explains the response. If the login attempt fails for any reason the ‘message’ field will explain why the failure occurred

Example

```
$baseService = "http://www.myhost.com/LiveTime/WebObjects/LiveTime.woa/ws/";

$objClient = new SoapClient($baseService . "Authenticate?wsdl", array('trace' =>
true));

$results = $objClient->connect($username, $password);

//This will return an Associative Array

$success = $results['success'];

if ($success=="false") {

    echo $results['message'] . "n";
```

```
    exit();
}
echo "Login Successfuln";
```

1.3.2 Disconnect

Syntax

disconnect()

Description

This API logs you out of the system.

Parameters

None

Return Value

This API returns a HashMap containing two fields: `success` and `message`.

The values returned in the two fields of the HashMap are:

1. The value returned in the `success` field will be a string representation of a boolean (‘true’ or ‘false’) indicating whether or not the attempt to log out was successful.
2. The value returned in the `message` field will be a text message that explains the response. If the request fails for any reason (i.e. the ‘success’ field is ‘false’), the ‘message’ field will explain why the failure occurred. Otherwise, you will see **Logged Out** displayed.

Example

```
function logout() {
    global $objClient;
    global $baseService;
    $objClient->SoapClient($baseService . "Authenticate?wsdl", array('trace' =>
true));
    $results = $objClient->disconnect();
    //This will return an Associative Array
    $success = $results['success'];
    if ($success=="false") {
        echo $results['message'] . "n";
        exit();
    }
    echo "Logout Successfuln";
```

```
    exit();
}
```

1.3.3 getLdapSources

Syntax

getLdapSources()

Description

This API returns a list of all LDAP sources when using the new multi-source authentication services in Service Desk.

Parameters

None

1.4 Customer API

1.4.1 Create Customer

Syntax

createCustomer (String username, String emailAddress, String firstname, String lastname, HashMap fieldValues)

Description

This API is used to create new customers in Service Desk.

If the Create Account option is enabled in the application’s setup, this function does not require the User to be logged in.

Parameters

The following are the input parameters for this API:

1. username: A new user name for this account.
2. emailAddress: Email address for this user.
3. firstname: First name of the user.
4. lastname: Last name of the user.
5. fieldValues: HashMap of the field values to populate this new account with.

NOTE: All four input parameters are mandatory and have to be unique. Omitting any of these will result in a failure message explaining the cause.

Return Value

This API returns a HashMap containing two fields: success and message.

The values returned in the two fields of the HashMap are:

1. The value returned in the success field will be a string representation of a boolean (true or false) indicating whether or not the Customer creation attempt was successful.
2. The value returned in the message field will be a String representation of the new Customer's ID number.

   NOTE: Make a note of the returned customer number for subsequent web service requests.

1.4.2 Find Customer

Syntax

findCustomer (String emailAddress)

findCustomer (String firstname, String lastname)

findCustomer (String emailAddress, String firstname, String lastname)

findCustomer (String username, String emailAddress, String firstname, String lastname, HashMap customFields)

Description

This API is used to search for a Customer.

Parameters

The following are the input parameters for this API:

1. Username: User name for this search. It can be either an LDAP user name or a System user name, which uses Service Desk's built-in authentication.
2. emailAddress: Email address for this search.

3. firstname: First name of user to search for.
4. lastname: Last name of user to search for.
5. customFields: HashMap of field values to locate this customer.

Return Value

This API returns a HashMap which contains a list of HashMap for each entry. The key of the first map is the Customer ID of the returned user, the HashMap for that entry contains four string fields representing the Customer's Last Name, Org Unit, Primary E-mail Address and First Name. A null is permitted when using multiple parameter based functions.

Example

```
function findUser($username,$email,$first,$last) {
    global $objClient;
    global $baseService;
    //switch to the customer api
    $objClient->SoapClient($baseService . "Customer?wsdl", array('trace' => true));
    echo "Searching for " . $username . $email . $first . $last . "n";
    # Call findCustomer, using an email address and echo out response
    # username, email, first, last
    $results = $objClient->findCustomer($username,$email,$first,$last);
    $success = $results['success'];
    if ($success=="false") {
        echo $results['message'] . "n";
        return;
    }
    # if we have results lets show them
    # returns an array of arrays
    foreach(array_values($results) as $ess) {
        print_r($ess);
    }}
```

1.4.3 Delete Customer

Syntax

deleteCustomerById (String customerId)

deleteCustomerByUsername (String username)

Description

This API is used to delete a Customer. The Customer ID or User name is required and the account will be locked out if the request is successful.

Parameters

The following are the input parameters for this API:

1. customerId: The unique customer ID of the account.
2. username: The unique user name of the account.

Return Value

This API returns a HashMap containing the success flag and a message if the request was unsuccessful.

1.4.4 Undelete Customer

Syntax

undeleteCustomerById (String customerId)

undeleteCustomerByUsername (String username)

Description

This API is used to undelete a customer. The Customer ID or user name is required and the account will be re- activated if the request is successful.

Parameters

The following are the input parameters for this API:

1. customerId: The unique customer ID of the account.
2. username: The unique user name of the account.

Return Value

This API returns a HashMap containing the success flag and a message if the request was unsuccessful.

1.4.5 Get Customer Information

Syntax

getFieldsForCustomer (String customerId)

Description

This API is used to get the Customer details to the logged in user, based on the supplied Customer ID.

Parameters

The following are the input parameters for this API:

1. customerId: The unique customer ID of the account.

Return Value

The returned HashMap contains fields that the system knows about the user like First Name, Last Name, Username and contact information.

Note that phone2 refers to the mobile/cell phone number in the LiveTime user interface, and phone3 refers to the fax number as defined in the GUI.

Custom Fields are defined in the administration area of Service desk. When retrieving these fields, they will be returned with the label defined in the custom field, but when calling update methods, these must be passed using the underlying variable name, being custom1 through custom5.

1.4.6 Get Country information

Syntax

getCountries ()

Description

This API is used to return a list of all the Countries.

Parameters

None

Return Value

This API returns a HashMap which contains another HashMap for each entry. The key of the first map is the Country ID of the returned Country, the Map for that entry contains two string fields representing the Country Name and Top Level Domain (TLD).

1.4.7 Get State Information

Syntax

getStatesForCountry (String countryId)

Description

This API is used to search for states or provinces of a Country.

Parameters

The following are the input parameters for this API:

1. countryId: The unique country ID.

Return Value

This API returns a HashMap containing the State ID and the State Name for each State of the searching Country.

1.4.8 Find Country

Syntax

findCountry (String countryName, String tld)

Description

This API is used to search for a Country, based on either the Country Name or Top Level Domain.

Parameters

The following are the input parameters for this API:

1. countryName: The unique country name for the search.
2. tld: The unique Top Level Domain used for the search.

Return Value

This API returns a HashMap containing another HashMap for the Country. The key of the first map is the Country ID of the returned Country, the Map for that entry contains two string fields representing the Country Name and Top Level Domain.

1.4.9 Update Customer

Syntax

updateCustomerById (String customerId, HashMap values)

updateCustomerByUsername (String username, HashMap values)

Description

This API allows logged in users to update Customer information.

Parameters

The following are the input parameters for this API:

1. customerId: The unique customer ID for the account.
2. username: The unique user name for the account.
3. values: The HashMap of values to set for this customer.

Return Value

The fields returned from the ‘get’ equivalent (firstname, lastname, email, address, city, state, etc) can all be inserted into the HashMap that is sent to LiveTime to set new values for those fields.

The returned HashMap will contain success and message values.

The values HashMap is a map of Strings to Strings. Valid keys for Customer web services are the same as those returned in the getFieldsForCustomer method, specifically: firstName, lastName, userName, password, email, orgUnit, address, addressTwo, city, state, country, postalCode, phone, phone2, phone3, webAccess, aliases, emailNotify.

All values should be strings, webAccess and emailNotify should be a 1 or a 0 expressed as a String, and aliases can be a comma-separated list.

1.4.10 Update Location

Syntax

updateLocation (String latitude, String longitude)

Description

This API allows logged in users to update their geographical coordinates by specifying latitude and longitude values.

Parameters

The following are the input parameters for this API:

1. latitude: The geographical latitude coordinate of the logged in account.
2. longitude: The geographical longitude coordinate of the logged in account.

Return Value

None

1.4.11 Get Geo Location

Syntax

getGeoLocation (String customerId)

Description

This API retrieves the geo location for the given customer ID.

Parameters

The following are the input parameters for this API:

1. customerId: The unique customer Id for the account.

Return Value

None

1.5 Item API

1.5.1 Create Item

Syntax

createItem (String itemTypeId, HashMap fieldValues)

createItem (String itemTypeId, String ownerCustomerId, HashMap fieldValues)

Description

This API enters a new Configuration Item (CI). Enter the Item Type ID, a Customer ID, and a set of Item Type field values.

The fieldValues HashMap is a Map of Strings to Strings. The keys represent Service desk data fields called custom1 through to custom20. These map to the category field definitions within the CMS. Other keys include itemNumber which allow the item to be created with a given item number (provided the Edit Item Numbers admin privilege is enabled).

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique Item number.

Return Value

This API returns a HashMap containing the Configuration Item ID and a success or failure message.

1.5.2 Delete Items

Syntax

deleteItem (String itemNumber)

Description

This API flags an Item as deleted so that it is no longer available for new requests.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique Item number.

Return Value

None

1.5.3 Undelete Items

Syntax

undeleteItem (String itemNumber)

Description

This API re-instates a previously deleted Item so that it is available for all future requests.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique Item number.

Return Value

None

1.5.4 Update Items

Syntax

updateItem (String itemNumber, HashMap fieldValues)

Description

This API updates the Item with new field values from the supplied hash map.

The fieldValues HashMap is a map of Strings to Strings. The keys represent Service desk data fields called custom1 through to custom20. Other keys include itemstatus and itemNumber for changes to the item status and item number fields respectively. These map to the Category field definitions within the CMDB.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique Item number.
2. fieldValues: A HashMap of field values for this Item.

Return Value

None

1.5.5 Get Items

Syntax

getItems ()

getItems (String customerId)

getItems (String customerId, String pageNdx)

getItems (String customerId, String itemNumber, HashMap fieldValues)

Description

This API returns a list of the item numbers that are either global or owned by the logged in Customer (or the Customer specified if using the customerId function). The response is in the form of a HashMap that uses the list of Item Numbers as field names. Each field has as its value a String containing the name of the Item Type that the Item represents.

In the event that no Items are found for the logged in Customer, only the ‘message’ field will be present and will contain a text message explaining that no Items were found.

Parameters

The following are the input parameters for this API:

1. customerId: The unique Customer Id.
2. itemNumber: The unique Item Number to retrieve. If this value is set, then fieldValues will be ignored.
3. fieldValues: A hash map of field names and values used to search for an appropriate CI. This will be ignored if the Item Number has already been set.

Return Value

The return value of this API is a HashMap containing a list of string values detailing the Item Number(s) and description(s). This returns the first 100 items. Use getItems(customerId, pageNdx) to page through larger result sets.

1.5.6 Get All Items

Syntax

getAllItems (String pageNdx)

Description

This API returns a list of Item Numbers for all the Items in the system. A page index is required to specify the page required. Each page contains 100 entries, ordered by Item Number.

Parameters

The following are the input parameters for this API:

1. pageNdx: The page index returns 100 entries ordered by Item Number.

Return Value

The return value of this API is a HashMap that uses the list of Item Numbers as field names. Each field has as its value a String containing the name of the Item Type that the Item represents.

1.5.7 Get Specific Item

Syntax

getSpecificItem (String itemNumber)

Description

This API returns the Item Type of the specified item number. This API provide a quick means to verify that a certain Item Number is defined in the system, without requiring heavy data exchange of Item attributes.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique Item number.

Return Value

None

1.5.8 Customer Items

Syntax

customerItems (String customerId, String pagesize, String pagenumber, String sortNdx)

customerItems (String pagesize, String pagenumber, String sortNdx)

Description

This API returns a list of the item numbers that are either owned by the customer or are global.

The response is in the form of a HashMap that uses the list of item numbers as field names. Each field has as its value a String containing the name of the Item Type that the Item represents.

In the argument (customerId) enter the ID of a Customer, this is assigned to the Customer when it is created via the createCustomer function.

Parameters

The following are the input parameters for this API:

1. customerId: A unique customer ID.
2. pagesize: The maximum number of items that can be returned per query.
3. pagenumber: In situations where there are more items than can fit on a pagesize, this number specifies the next set of items to be retrieved.
4. sortNdx: The index of the column to sort on. The possible values for the parameter are:
    a. 1: Item Number
    b. 2: Item Category
    c. 3: Item Type
    d. 7: Item Status
    e. 8: Purchase Date
    f. 9: Incident Team
    g. In Service Manager, you can also specify:
        i. 10: Problem Team
        ii. 11: Change Team
        iii. 12: Service Request Team

Return Value

This API returns a HashMap containing a list of string values detailing the Item Number(s) and description(s), being the corresponding Item Types.

In the event that no Items are found for the Customer, only the message field will be present and will contain a text message explaining that no Items were found.

1.5.9 Org Unit Items

Syntax

orgUnitItems (String orgUnitId, String pagesize, String pagenumber, String sortNdx)

orgUnitItems (String pagesize, String pagenumber, String sortNdx)

Description

This API is used to search for Items owned by the logged in User's Org Unit or the Org Unit ID specified.

Parameters

The following are the input parameters for this API:

1. orgUnitId: A unique Org Unit ID.
2. pagesize: The maximum number of items that can be returned per query. The page size must be between 10 and 100.

3. pagenumber: In situations where there are more items than can fit on a pagesize, this number specifies the next set of items to be retrieved. The page number must be at least 0.
4. sortNdx: The index of the column to sort on. The possible values for the parameter are:
    a. 1: Item Number
    b. 2: Item Category
    c. 3: Item Type
    d. 7: Item Status
    e. 8: Purchase Date
    f. 9: Incident Team
    g. In Service Manager, you can also specify:
        i. 11: Problem Team
        ii. 12: Change Team

Return Value

This API returns a HashMap containing of HashMaps. The keys of the first map are the Item Numbers. Each Item Number has its own HashMap representing the values that represent Item attributes as name-value pairs.

1.5.10 Add Item Owner

Syntax

addItemOwner (String itemNumber, String username)

Description

This API adds a new Customer from an Item with the specified item number.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique Item number.
2. username The user name of the user who wants to add the item owner.

Return Value

None

1.5.11 Remove Item Owner

Syntax

removeItemOwner (String itemNumber, String username)

Description

This API removes an existing Customer from an Item with the specified item number.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique Item number.
2. username The user name of the user who wants to remove the item owner.

Return Value

None

1.5.12 Get Item Owner

Syntax

getItemOwners (String itemNumber)

Description

This API returns a list of all the owners of an item with the specified item number.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique Item number.

Return Value

This API returns a HashMap which contains another HashMap for each entry. The key of the first map is the Customer ID of the returned user, the HashMap for that entry contains four string fields representing the Customer's Last Name, Org Unit, Primary E-mail Address and First Name. Global items will return a success response message.

1.5.13 Get Fields For Item

Syntax

getFieldsForItem (String itemNumber)

Description

This API returns a list of all Configuration Item fields for the specified item number.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique Item number.

Return Value

This API returns two HashMaps, the first containing a list of CI field Ids, and the second containing the field values include the field name, whether the field is required, the data type, the field's default value (if specified), and the field options if any are available.

1.5.14 Get Fields For Item Type

Syntax

getFieldsForItemType (String itemTypeId)

Description

This API returns a list of all Configuration Item fields for the specified Item Type ID.

Parameters

The following are the input parameters for this API:

1. itemTypeId: The unique Item type id.

Return Value

This API returns two HashMaps, the first containing a list of CI field Ids, and the second containing the field values include the field name, whether the field is required, the data type, the field's default value (if specified), and the field options if any are available.

1.5.15 Get Fields For Related Item

Syntax

getFieldsForRelatedItem (String itemNumber)

Description

This API returns information on all relationships for the specified item number.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique Item number.

Return Value

This API returns a HashMap containing of HashMaps, the outer keyed on the Item Numbers – the HashMap for each Item contains data such as the related Item Number, Criticality, Type and how it's related to the Item with the given itemNumber.

1.5.16 Create Manufacturer

Syntax

createManufacturer (String name)

Description

This API creates a Manufacturer with the given name.

Parameters

The following are the input parameters for this API:

1. name: The unique name of the Manufacturer to be created.

Return Value

None

1.5.17 Add Item Note

Syntax

addItemNote(String itemNumber, String note)

Description

This API adds a new note to an item with the give itemNumber.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique Item Number of the Item to be retrieved.
2. note: The note contents

Return Value

None

1.5.18 Update Item Description

Syntax

updateItemDescription(String itemNumber, String desc)

Description

This API updates the description of an item with the given itemNumber. If the item does not have any description then it will add description to the item.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique Item Number of the Item to be retrieved.
2. desc: The description of the contents

Return Value

None

1.5.19 Update Item Cost

Syntax

updateItemCost(String itemNumber, String cost)

Description

This API updates the cost of an item with the given itemNumber. If the item does not have any cost then it will add cost to the item.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique Item Number of the Item to be retrieved.
2. cost: The cost of the item to be updated

Return Value

None

1.5.20 Get Item Categories

Syntax

getItemCategories ()

Description

This API returns a list of all the Configuration Item (CI) Categories defined in the system.

Parameters

None

Return Value

This API returns a HashMap containing a list of string values including the Category ID and the corresponding string value.

1.5.21 Get Item Type For Category

Syntax

getItemTypesForCategory (String categoryId)

Description

This API returns a list of all the Item Types that exist in the system for the specified Category ID.

Parameters

The following are the input parameters for this API:

1. categoryId: The unique category Id for which item type has to be retrieved.

Return Value

This API returns a HashMap containing a list of string values including the Item Type ID and the corresponding string value.

1.5.22 Create Item Type

Syntax

createItemType (String itemTypeName, String incidentTeamID, String problemTeamID, String changeTeamID, String requestTeamID, String manufacturerName, String categoryID)

Description

This API creates an ItemType (product) based on the name, the provided support teams, the manufacturer (which will be created if it doesn't exist) and the Category ID which is returned by getItemCategories().

Parameters

The following are the input parameters for this API:

1. itemTypeName: A name for the Item Type
2. incidentTeamID: The default Incident Team Id for this Item Type
3. problemTeamID: The default Problem Team Id for this Item Type
4. changeTeamID: The default Change Team Id for this Item Type
5. requestTeamID: The default Service Request Team Id for this Item Type
6. manufacturerName:The name of Manufacturer (created if not available)
7. categoryId: The default categoryId to associate the new Item Type

Return Value

None

1.5.23 Get Item Type

Syntax

getItemType (String itemTypeId)

Description

This API returns item type details of the specified item type id. This API exists as a quick means to determine what category an item type is defined as.

Parameters

The following are the input parameters for this API:

1. itemTypeId: The unique item type Id for which details have to be retrieved.

Return Value

This API returns value is a HashMap containing the item type name and the category name.

1.5.24 Get Item Type

Syntax

getItemType (String itemTypeId)

Description

This API returns item type details of the specified item type id. This API exists as a quick means to determine what category an item type is defined as.

Parameters

The following are the input parameters for this API:

1. itemTypeId: The unique item type Id for which details have to be retrieved.

Return Value

This API returns value is a HashMap containing the item type name and the category name.

1.5.25 Get Item Relationship Type

Syntax

getItemRelationshipTypes()

Description

This API returns all the relationships, along with their class and a marker to denote inherited ownership.

Parameters

None

Return Value

The return value format is *RelationName [RelationClass] (the asterix will not be present if ownership is not inherited). The Key to the HashMap is the relationTypeId, which needs to be passed into the createItemRelationship() function (below).

1.5.26 Create Item Relationships

Syntax

createItemRelationship(String fromItemId, String toItemId, String relationTypeId)

Description

This API is used to create a new relationship if it does not exist.

Parameters

The following are the input parameters for this API:

1. fromItemId: The unique Id of the source Item to create a relationship with parent
2. toItemId: The unique Id of the child Item to create the relationship
3. relationTypeId: The Id returned by getItemRelationshipTypes() function signifying the type of relationship to create

Return Value

None

1.5.27 Delete Item Relationships

Syntax

deleteItemRelationship (String fromItemId, String toItemId, String relationTypeId)

Description

This API is used to delete the relationship.

Parameters

The following are the input parameters for this API:

1. fromItemId: The unique Id of the source Item
2. toItemId: The unique Id of the child Item
3. relationTypeId: The Id returned by getItemRelationshipTypes() function signifying the type of relationship

Return Value

None

1.5.28 Get Item Relationships

Syntax

getItemRelationships (String itemNumber)

Description

This API returns all the related items for the given itemNumber.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique item number to identify the relationships for.

Return Value

This API returns a HashMap containing the Item Number of the related Item and a Relationship ID.

1.5.29 Get Incident Teams

Syntax

getIncidentTeams()

Description

This API returns the Teams for the Incident process (1000).

Parameters

None

Return Value

None

1.5.30 Get Problem Teams

Syntax

getProblemTeams()

Description

This API returns the Teams for the Problem process (2000).

Parameters

None

Return Value

None

1.5.31 Get Change Teams

Syntax

getChangeTeams()

Description

This API returns the Teams for the Change process (3000).

Parameters

None

Return Value

None

1.5.32 Get Service Request Teams

Syntax

getServiceRequestTeams()

Description

This API returns the Teams for the Service Request process (7000).

Parameters

None

Return Value

None

1.6 Request API

1.6.1 Create Incident / Change Request / Service Request

Syntax

createIncident (String itemNumber, String classificationId, String subject, String description, HashMap customFieldValues)

createChangeRequest (String itemNumber, String classificationId, String subject, String description, HashMap customFieldValues)

createServiceRequest (String itemNumber, String classificationId, String subject, String description, HashMap customFieldValues)

Description

This API is used to create an Incident, Change Request or Service Request for the logged in User. Simply supply the function with the number of the Item you wish to raise an Incident, Change or Service Request against, the ID of the Classification the Incident falls under and a text description of the problem.

If custom fields are enabled for the Request type being created, these values need to be passed into the customFieldValues HashMap, with keys custom1 through to custom5.

The Item Number and Classification must exist in this function’s derived lists for the Request to be successfully created.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique Item Number used for the basis of this request.
2. classificationId: The Item classification Id to use for this request. Use getClassifications(String itemNumber) for a list of classifications available for this item.
3. subject: The subject line of this new request.
4. description: The description of this request.
5. customFieldValues: A HashMap of values with keys custom1 through custom5.

NOTE: All three arguments ('itemNumber,' 'classificationNumber' and 'Description') are required. Omitting any of the three will result in failure and a message explaining the circumstances.

Return Value

This API returns a HashMap containing two fields: success and message.

The value returned in the success field will be a string representation of a boolean (true or false) indicating whether or not the Request's creation attempt was successful.

If a Request was successfully created (that is the success field is true), the value returned i the message field will be a String representation of the new Request's ID number.

If the process fails for any reason (that is. the success field is false), the message field will contain a text message explaining why the failure occurred.

1.6.2 Create Incident / Change Request / Service Request (for a Customer)

Syntax

createIncidentCustomer (String customerId, String itemNumber, String classificationId, String subject, String description, HashMap customFieldValues)

createIncidentDates(String customerId, String itemNumber, String classificationId, String subject, String description, HashMap customFieldValues, String reportDate, String closeDate, String closingTechId)

createChangeRequestCustomer (String customerId, String itemNumber, String classificationId, String subject, String description, HashMap customFieldValues)

createServiceRequestCustomer (String customerId, String itemNumber, String classificationId, String subject, String description, HashMap customFieldValues)

Description

This API is used to create an Incident, Change Request or Service Request for a Customer created from the createCustomer() function. Provide the Customer ID, the Item Number you wish to raise the Request against, the ID of the Classification the Request falls under and a text description of the problem.

If custom fields are enabled for the Request type being created, these values need to be passed into the customFieldValues HashMap, with keys custom1 through to custom5.

The Item Number and Classification must exist in this function's derived lists for the Request to be successfully created.

Parameters

The following are the input parameters for this API:

1. customerId: The unique Customer Id to create the request against.
2. itemNumber: The unique Item Number used for the basis of this request.
3. classificationId: The Item classification Id to use for this request. Use getClassifications(String itemNumber) for a list of classifications available for this item.
4. subject: The subject line of this new request.
5. description: The description of this request.
6. customFieldValues: A HashMap of values with keys custom1 through custom5.
7. reportDate: The date the request will be initially recorded. Date must be formatted as yyyy-mm-dd hh:mm.
8. closeDate: The date the request should be recorded as closed. Date must be formatted as yyyy-mm-dd hh:mm.
9. closingTechId: The id of the technician that closed the request. The closingTechId if set will be used in conjunction with the closeDate to set the technician to credit for the closure.

NOTE: All four arguments ('customerId', 'itemNumber,' 'classificationNumber' and 'Description') are required.

reportDate and closeDate must be formatted as 'yyyy-mm-dd hh:mm' strings to be correctly parsed and will set the dates the request was logged and closed, setting the appropriate times against the SLA. This can be combined with adding incident notes to import legacy data into the system.

Return Value

This API returns a HashMap containing two fields: success and message.

The value returned in the success field will be a string representation of a boolean (true or false) indicating whether or not the Request's creation attempt was successful.

If a Request was successfully created (that is the success field is true), the value returned i the message field will be a String representation of the new Incidents's ID number.

If the process fails for any reason (that is. the success field is false), the message field will contain a text message explaining why the failure occurred.

1.6.3 Get Classifications

Syntax

getClassifications (String itemNumber)

Description

This API returns a list of all the classifications that exist in the system for the selected item type.

Parameters

The following are the input parameters for this API:

1. itemNumber: The unique ItemNumber to retrieve classifications from.

Return Value

This API returns a HashMap that contains the list of Classification IDs as field names. Each field has a String value containing the name of the Classification. In Service Desk, Classifications are grouped into a hierarchical tree. The list returned by ‘getClassifications’ will be ordered according to the hierarchical tree as it would appear in the application.

In the event that no Classifications are found, only the ‘message’ field will be present and will contain a text message explaining that no Classifications were found.

Once you have the names of the fields (iter.next() – Classification ID numbers in the above example) you can then use them to look up the names of the Classifications associated with them in the HashMap.

1.6.4 Get Subject

Syntax

getSubject (String requestId)

Description

This API returns a HashMap containing the ‘subject’ for the provided Request ID.

Parameters

The following are the input parameters for this API:

1. requestId: The unique id of the request to retrieve subject from.

Return Value

None

1.6.5 Get Description

Syntax

getDescription (String requestId)

Description

This API returns a HashMap containing the ‘description’ for the provided Request ID.

Parameters

The following are the input parameters for this API:

1. requestId: The unique of the request to retrieve description from.

Return Value

None

1.6.6 Get Request Count By Status

Syntax

getRequestCountByStatus (String statusId)

Description

This API returns a HashMap containing the ‘requestCount’ for the provided Status ID.

Parameters

The following are the input parameters for this API:

1. statusId: The id of the status to retrieve the request count for. This is the a synonym for the workflow status.

Return Value

None

1.6.7 Get Requests By Status

Syntax

getRequestsByStatus (String statusId, String pageNo)

Description

This API returns a HashMap containing the ‘requestId’ as a key and the description as the value, for all Requests in a Status with the given statusId. The returned HashMap also contains the success flag to deal with Requests that can’t be processed for various reasons, along with a message to state the reason.

Parameters

The following are the input parameters for this API:

1. statusId: The id of the status to return requests against.
2. pageNo: The page number to retrieve. These are returned 100 requests at a time.

Return Value

None

1.6.8 Get Request Detail

Syntax

getRequestDetail (String requestId)

Description

This API returns a HashMap containing various fields that make up the request. The returned HashMap also contains the success flag to deal with requests that can’t be processed for various reasons, along with a message to state the reason.

Parameters

The following are the input parameters for this API:

1. requestId: The unique request Id of the request to return details from.

Return Value

The API returns a HashMap contains a success flag and the values for the keys of the given fields:

Key	Description
customer	Customers full name
customerId	Customer ID
customerPhone	Customers contact phone number.
orgUnit	Organizational Unit display name
orgUnitId	Organizational Unit ID
item	Configuration Item name

Key	Description
itemId	Configuration Item ID
itemNumber	Configuration Item Number as displayed to users.
itemStatus	Configuration Item status.
itemCriticality	Configuration Item criticality.
status	Request status
statusId	Request status ID
statusEntry	If the request is at an entry state of the workflow
statusExit	If the request is at an exit state of the workflow
statusIsApproval	If the request is in an approval status state on the workflow
classification	Request classification
classificationId	Request classification ID
impact	Request impact
impactId	Request impact ID
urgency	Request urgency
urgencyId	Request urgency ID
priorityType	Request priority
priorityTypeId	Request priority ID
technician	Assigned Technician
team	Assigned Team
teamId	Assigned Team ID
dateCreated	Request creation date (MM/dd/yyyy HH:mm)
dateClosed	Request closed date (MM/dd/yyyy HH:mm)
dateDue	Request due date (MM/dd/yyyy HH:mm)
lastUpdated	Date request was last updated (MM/dd/yyyy HH:mm)
type	Process type
workflow	The current workflow
workflowId	The current workflow Id
editable	If the request is editable by the current user based on team membership and other permissions

1.6.9 Get Public Notes

Syntax

getPublicNotes (String requestId)

Description

This API returns a list of all the public Notes for a given requestId. The returned HashMap contains the Note ID and Note Date.

Parameters

The following are the input parameters for this API:

1. requestId: The unique Request Id from which to retrieve notes.

Return Value

None

1.6.10 Get Notes

Syntax

getNotes (String requestId)

Description

This API returns a list of all the Notes for a given requestId. The returned HashMap contains the Note ID and Note Date.

Parameters

The following are the input parameters for this API:

1. requestId: The unique Request Id from which to retrieve notes.

Return Value

None

1.6.11 Get Notes Detail

Syntax

getNotesDetail (String requestId, int pageSize, int pageNumber)

Description

This API returns a list of all the Notes for a given requestId. The page size and page number can also be specified. The returned HashMap contains the Note ID, Note Date, Private Note Flag, Author and the Note Content.

Parameters

The following are the input parameters for this API:

1. requestId: The unique Request Id from which to retrieve notes.
2. pageSize: The number of elements to return for each page.
3. pageNumber: The page number to retrieve with given pageSize.

Return Value

None

1.6.12 Get Note Content

Syntax

getNoteContent (String requestId, String noteId)

Description

This API returns the content of the Note for the provided Request ID and Note ID.

Parameters

The following are the input parameters for this API:

1. requestId: The unique Request Id from which to retrieve note content.
2. noteId: The note id to retrieve.

Return Value

None

1.6.13 Update Request

Syntax

updateRequest (String requestId, HashMap fieldValues)

Description

This API is used to update a request of any type (Incident, Change Request or Service Request).

Parameters

The following are the input parameters for this API:

1. requestId: The Request Id to be updated
2. fieldValues: The Map of Strings to Strings. Valid keys are: customerId, itemId, statusId, classificationId, impactId, urgencyId, custom1, custom2, custom3, custom4 and custom5. (Since 6.1) typeCustom1, typeCustom2, typeCustom3, typeCustom4 and typeCustom5.

Return Value

This API returns a HashMap containing a success flag to deal with Requests that can not be processed for various reasons, along with a message to state the reason.

1.6.14 Find Incident / Change Request / Service Request

Syntax

findIncident (String itemNumber, String classificationId, HashMap fieldValues)

findChangeRequest (String itemNumber, String classificationId, HashMap fieldValues)

findServiceRequest (String itemNumber, String classificationId, HashMap fieldValues)

Description

This API is used to find an Incident, Change Request or Service Request. Simply supply the function with the parameters desired to search against and process the resultant list of requests that are returned. The ‘fieldValues’ parameter can take all the arguments that are used in the updateRequestDetail method above.

Parameters

The following are the input parameters for this API:

1. itemNumber: Find and Incident based upon on a unique item number
2. classificationId: Find an incident based on this classification Id.
3. fieldValues: A haspmap of fieldValues that can be searched. Takes the same keys as updateRequestDetail.

Return Value

This API returns a HashMap containing the success flag and a message.

1.6.15 Add Note To Request

Syntax

addNoteToRequest (String requestId, String note)

addNoteToRequest (String requestId, String note, String privateNdx)

addNoteToRequest (String requestId, String note, String privateNdx, String timeMins)

Description

This API is used to add a note to a request by members of the assigned team, or the customer of the request. If the privateNdx flag is specified, the value should either be set to 1 or 0, to define the note as being private or public respectively. Time can be assigned to this note for timesheet reporting by using the associated method and adding the time in minutes.

Parameters

The following are the input parameters for this API:

1. requestId: The requestId to add the note to.
2. note: The note contents.
3. priateNdx: A flag to define the note as being private (1) or public (0).
4. timeMins: The time in minutes to add to this note for timesheet reporting and billing.

Return Value

This API returns a HashMap containing the success flag to deal with requests that can’t be processed for various reasons, along with a message to state the reason. If privateNdx is not specified, added notes will be visible to the public.

1.6.16 Close Request

Syntax

closeRequest (String requestId, String reason)

Description

This API is used to set the Request to the default closed state of the assigned Workflow and add the supplied reason as the solution (if defined), and the target Request is an Incident or Service Request.

Parameters

The following are the input parameters for this API:

1. requestId: The request Id to be closed.
2. reason: The reason for closing this request.

Return Value

This API returns a HashMap containing the success flag and a reason field.

1.6.17 Get Response Templates

Syntax

getResponseTemplates (String requestId)

Description

This API returns a list of valid response templates for this request.

Parameters

The following are the input parameters for this API:

1. requestId: The request Id to retrieve the response templates for.

Return Value

This API returns the responseId as the key and the template Title as the value in the response. Use getResponseTemplateContent to retrieve the actual content of a response based on a returned ID from this call.

1.6.18 Get Response Templates Content

Syntax

getResponseTemplateContent (String responseId)

Description

This API returns the actual contents of a response template by ID.

Parameters

The following are the input parameters for this API:

1. responseId: The response Id of the template to retrieve.

Return Value

Use getResponseTemplates to get a list of valid ID’s you can use for this call. Note that the text will be HTML formatted if the text was styled using the rich text editor.

1.6.19 Get My Filters

Syntax

getMyFilters ()

Description

This API returns a list of Filters by Id and name for the current logged in user.

Parameters

None

Return Value

This API returns a HashMap containing the filter Id as the key and filter name as the value.

1.6.20 Get My Task Count

Syntax

getMyTaskCount ()

getMyTaskCount (int filterId)

Description

This API retrieves the number of currently active and assigned tasks to the current logged in user. If a filterId is passed in then it will return the number of requests with the given filter applied.

Parameters

The following are the input parameters for this API:

1. filterId: The unique filter Id to use to return the number of requests available with that filter.

Return Value

This API returns a HashMap containing a success flag and the number of tasks.

1.6.21 Get My Tasks

Syntax

getMyTasks (int pageSize, int pageNumber, List fields, HashMap sortMap)

getMyTasks (int filterId, int pageSize, int pageNumber, List fields, HashMap sortMap)

Description

This API retrieves a list of tasks based on the data sent to the web-service function. The returned HashMap contains a success flag and the values for the keys of the given fields.

Requests will be sorted by their ID in ascending order if no sortMap is defined. The sortMap accepts all the above fields as keys. Acceptable values include:

1. ASC – perform an ascending sort on the field
2. DESC- perform a descending sort on the field

Key	Description
customer	Customers full name
orgUnit	Organizational Unit display name
item	Item number
status	Status name
classification	Classification
impact	Impact name
urgency	Urgency name
priorityType	Priority name
team	Team name
type	Process type
status	status name
dateCreated	Request creation date (MM/dd/yyyy HH:mm)
dateClosed	Request closed date (MM/dd/yyyy HH:mm)
dateDue	Request due date (MM/dd/yyyy HH:mm)
lastUpdated	Date request was last updated (MM/dd/yyyy HH:mm)
subject	Request subject
description	Request description
custom1…custom5	Request custom fields
typeCustom1…typeCustom5	Item type custom fields

Parameters

The following are the input parameters for this API:

1. filterId: The filterId to apply when returning the tasks.
2. pageSize: The maximum number of tasks that can be returned per query
3. pageNumber In situations where there are more tasks than can fit on a page size, this number specifies the next set of tasks to be retrieved.
4. fields Specifies the field data whose values will be retrieved from the task.
5. sortMap Specifies the fields on which to sort the retrieved tasks.

Return Value

This API returns a HashMap containing a success flag and the number of tasks.

1.6.22 Get Attachments

Syntax

getAttachments (String requestId)

Description

This API retrieves a list of attachments for the defined requestId as a Name/URL as a key/value pair. As long as the session is valid you can then stream the file directly to the client using the specified URL.

Parameters

The following are the input parameters for this API:

1. requestId: The request Id for which attachments have to be got.

Return Value

None

1.6.23 Get Incident States

Syntax

getIncidentStates ()

Description

This API returns all Workflow States allowed for Incidents in a HashMap, keyed on ID and name formatted as ‘workflow – state name’.

Parameters

None

Return Value

None

1.6.24 Get Problem States

Syntax

getProblemStates ()

Description

This API returns all Workflow States allowed for Problems in a HashMap, keyed on ID and name formatted as ‘workflow – state name’.

Parameters

None

Return Value

None

1.6.25 Get Change States

Syntax

getChangeRequestStates ()

Description

This API returns all Workflow States allowed for Change Requests in a HashMap, keyed on ID and name formatted as ‘workflow – state name’.

Parameters

None

Return Value

None

1.6.26 Get Service Request States

Syntax

getServiceRequestStates ()

Description

This API returns all Workflow States allowed for Service Requests in a HashMap, keyed on ID and name formatted as ‘workflow – state name’.

Parameters

None

Return Value

None

1.6.27 Get Next States

Syntax

getNextStates (String requestId)

Description

This API returns the next available Workflow States allowed for the identified request in a HashMap, keyed on ID and name formatted as ‘state name’.

Parameters

The following are the input parameters for this API:

1. requestId: The request Id for which attachments have to be got.

Return Value

None

1.7 Organization API

Organizational Units are defined as a two tier structure, reflecting a company-department structure. The following API’s require the user to pass the Org Unit Name they are editing, along with the parent Org Unit. An empty string can be provided when a top level (Company) Org Unit is being created and/or edited.

1.7.1 Create Org Unit

Syntax

createOrgUnit (String name, String parent, HashMap fieldValues)

Description

This API is used to create an Organizational Unit. The Org Unit Name, the parent Org Unit Name and a map of field values are the inputs.

If the Org Unit being created is a company, pass an empty string as the parent field. If creating a department (or child org unit) the parent must already exist or an error message will be returned.

The complete list of fields that can be supplied reflect the fields returned from the get method below – but names and contact information are the general values, along with any custom fields that may be defined.

Parameters

The following are the input parameters for this API:

1. name: The name of the Org unit to be created.
2. parent: The name of the parent if present.
3. fieldValues: The map of field values.

Return Value

This API returns a Map containing the success field and a message field if the method call failed.

1.7.2 Get Org Unit Details

Syntax

getOrgUnitDetails (String name, String parent)

getOrgUnitDetails (String orgUnitId)

Description

This API is used to get details of an Organizational Unit. The Org Unit id or Org Unit Name and a map of field values are the inputs.

Parameters

The following are the input parameters for this API:

1. name: The name of the Org unit.
2. parent: The name of the parent of the Org Unit if there are any.
3. orgUnitId: The Org Unit Id to retrieve details from.

Return Value

This API returns a HashMap of name value pairs representing the fields of the company or department. The valid keys for organization web services are: address, addressTwo, city, state, country, zip, phone, url, primaryContact.

The returned map will also contain a success field. If there was a problem with the execution there will be a message field to explain what the issue was.

1.7.3 Update Org Unit Details

Syntax

updateOrgUnit (String name, String parent, HashMap fieldValues)

Description

This API updates the organizational unit and a parameter map containing the fields to update and the new values for those fields.

Parameters

The following are the input parameters for this API:

1. name: The name of the Org unit
2. parent: The name of the parent of the Org Unit if there are any
3. fieldValues: The map of the field values.

Return Value

This API returns a HashMap containing the success field and a message field if the method call failed.

1.7.4 Rename Org Unit

Syntax

renameOrgUnit (String name, String parent, String newName)

Description

This API renames the organizational unit with the new name. The two strings name and parent are used to identify the organizational unit where a parent value of an empty string defines a company, or content in both fields defining a department.

Parameters

The following are the input parameters for this API:

1. name: The name of the Org unit

2. parent: The name of the parent of the Org Unit if there are any
3. newname: The new name of the Org unit.

Return Value

This API returns a HashMap containing the success field and a message field if the method call failed.

1.7.5 Find Org Unit

Syntax

findOrgUnit (String name, String parent, HashMap fieldValues)

Description

This API is used to search for an org unit. Valid keys for the map include ‘name’ and ‘parent’ along with all the parameters defined in the org unit details method: address, addressTwo, city, state, country, zip, phone, url, primaryContact.

Parameters

The following are the input parameters for this API:

1. name: The name of the Org unit
2. parent: The name of the parent of the Org Unit if there are any
3. fieldValues: The map of the field values.

Return Value

This API returns a HashMap containing the ID’s and the display strings of the org units that match the criteria along with the success field and a message field if the method call failed.

1.7.6 Delete Org Unit

Syntax

deleteOrgUnit (String orgUnitId)

Description

This API flags an Org Unit as deleted.

Parameters

The following are the input parameters for this API:

1. orgUnitId: The Org unit id to be deleted.

Return Value

None

1.8 Purchase Order API

1.8.1 Create Purchase Order

Syntax

createPurchaseOrder (String vendorName, String originatorId, String deliverToId, HashMap fieldValues, HashMap lineItems)

Description

This API is used to create a new purchase order in the financial module.

Parameters

The following are the input parameters for this API:

1. vendorName: The name of the vendor to create purchase order against.
2. originatorId: The Id of the client originating the purchase order.
3. deliverToId: The Id of the client receiving the order.
4. fieldValues: The map of field values which takes key custom1…custom5 and their associated values
5. lineItems: The map of field values which takes keys and values, itemtype (id), partnumber, quantity (optional, defaults to 1), price.

Return Value

None

1.8.2 Create Lease Purchase Order

Syntax

createcreateLeasePurchaseOrder PurchaseOrder (String vendorName, String originatorId, String deliverToId, HashMap fieldValues, HashMap lineItems, String leaseDurationId)

Description

This API is used to create a new lease purchase order in the financial module.

Parameters

The following are the input parameters for this API:

1. vendorName: The name of the vendor to create purchase order against.
2. originatorId: The Id of the client originating the purchase order.
3. deliverToId: The Id of the client receiving the order.
4. fieldValues: The map of field values which takes key custom1…custom5 and their associated values
5. lineItems: The map of field values which takes keys and values, itemtype (id), partnumber, quantity (optional, defaults to 1), price, leaseDurationId.

Return Value

None

1.8.3 Deliver Purchase Order

Syntax

deliverPurchaseOrder (String poNumber)

Description

This API is used to confirm delivery of a purchase order in the system.

Parameters

The following are the input parameters for this API:

1. poNumber: The number of the purchase order.

Return Value

None

1.8.4 Get Fields For Purchase Order

Syntax

getFieldsForPurchaseOrder (String poNumber)

Description

This API returns the fields of a purchase order in the system.

Parameters

The following are the input parameters for this API:

1. poNumber: The number of the purchase order.

Return Value

None

1.8.5 Delete Purchase Order

Syntax

deletePurchaseOrder (String poNumber)

Description

This API is used to delete a purchase order.

Parameters

The following are the input parameters for this API:

1. poNumber: The number of the purchase order.

Return Value

None

1.9 Localizing Service Desk ITSM

Internationalization is the process of designing an application so that it can be adapted to various languages and regions without engineering changes. Sometimes the term internationalization is abbreviated as i18n, because there are 18 letters between the first i and the last n.

The Service Desk product suite is an internationalized program that has the following characteristics:

- With the addition of localized data, the same executable can run worldwide.
- Textual elements, such as status messages and the GUI component labels, are not hard coded in the program. Instead they are stored outside the source code and retrieved dynamically.
- Support for new languages does not require re-compilation.
- Culturally-dependent data, such as dates and currencies, appear in formats that conform to the end user's region and language.
- Service Desk can be localized quickly.

Localization is the process of adapting software for a specific region or language by adding locale-specific components and translating text. The term localization is often abbreviated as l10n, because there are 10 letters between the l and the n.

The Service Desk internationalized framework is very intuitive and flexible, allowing Users to localize it for a particular language and character encoding scheme.

1.9.1 Getting Started

The primary task of Users performing localization of the Service Desk application is to translate the user interface elements in the external text file – LiveTime.properties.

The following steps guide you through the localization process. We will assume a French localization for this example, which uses the ISO-8859-1 character set.

1. Ensure you have the appropriate character set installed on your Desktop’s Operating System. The localization file and rendering will both require the character set to be available for data entry and data presentation.
2. Ensure your RDBMS has been created using the appropriate character set, usually UTF-8. This is required so that French data entry can be successfully saved and retrieved from the database.
3. Specify the correct character set encoding for Service Desk. Edit the ‘Properties’ file (/CONTAINER_APPS_PATH>/LiveTime/WEB-INF/LiveTime.woa/Contents/Resources folder) and specify the character set to use for presentation to the end user. This character set will be sent to the end users to control presentation in both the user interface and outbound email.

1.9.2 Customizing Strings

The LiveTime.properties file (/CONTAINER_APPS_PATH>/LiveTime/WEB-INF/LiveTime.woa/Contents/Resources folder) stores the externalized user interface content. This file is stored in plain text. The names of the localized files do follow a standard. In our example of French, livetime_fr.properties was the resultant file name. The fr component of this name comes from iso 639-1. The default (English) is encoded using the ISO-8859-1 character set. You can create alternative versions of this file in any text editor, that allows the file to be saved in the necessary character set.

The first step in creating a local version of this file is to save a copy of it representing the appropriate language. Being a French translation, there is no need to worry about the file encoding, so simply save a copy of the file called livetime_fr.properties.

1.9.3 Simple Messages

The copy of this file consists of a collection of name-value pairs, each are broken up into clusters representing user interface components. For example:

```
ClassName.greetings=Hello

ClassName.farewell=Goodbye

ClassName.inquiry=How are you?
```

The name or key section (ClassName.greetings) of these strings is of little relevance to the end user. They are simply markers inside the application representing where each string goes.

The value component of these pairs (Hello) represents the corresponding user interface value. The (French) translator can safely edit these values and they will appear within Service Desk.

```
ClassName.greetings=Bonjour.

ClassName.farewell=Au revoir.

ClassName.inquiry=Comment allez-vous?
```

1.9.4 Compound Messages

Compound messages contain variable data – that is data that Service Desk will substitute into the message at run time. For example, given the message Hello Sunshine, the string Sunshine may vary. This message is difficult to translate because the position of the string in the sentence is not the same in all languages, or the value may need to be generated by Service Desk based on some business logic.

Where this substitution is necessary, the localized strings will have a set of braces containing an id number (a substitution variable). These blocks will be replaced at run time by Service Desk and are identified by their presence. The localized version of a given String must contain the same number of substitution variables.

For example:

```
ClassName.helloSunshine=hello {0}
```

This will render as hello sunshine?however we could modify this message to read sunshine, hello by doing this:

```
ClassName.helloSunshine={0}, hello
```

1.9.5 Locales

The LiveTimeLocales.xml file (/CONTAINER_APPS_PATH>/LiveTime/WEB-INF/LiveTime.woa/Contents/Resources folder) defines the language code and charset used in order to process the strings defined in the external properties file.

Typically, this will be the charset used when saving the external properties file. By default the following language locale is selected:

```
<LOCALE language="en" country="" charset="ISO-8859-1"/>
```

To enable our French external properties file to behave correctly, we will need to uncomment the following line:

```
<LOCALE language="fr" country="" charset="UTF-8"/>
```

This will enable the application to use both English and French based on the user's browser and operating system settings. For convenience, many values are already supplied in this file, you need to uncomment the necessary entry.

If an entry for your locale language doesn’t exist, then you can define it in this file by supplying the language code, country code, and charset, as illustrated in the above examples.

Instructions for setting browser locales is outside the scope of this document as each browser/OS combo handles this differently.

1.10 SOA Guide

A Service-oriented architecture (SOA) is software built using loosely coupled software services to support the requirements of business processes and software users. Network resources in a SOA environment are made available as independent services that can be accessed irrespective of the underlying platform implementation.

More importantly, SOA architecture enables the creation of applications that are built by combining loosely coupled and interoperable services. These services inter-operate based on a formal definition (or contract, e.g., WSDL) that is independent of the underlying platform and programming language, and the interface definition hides the implementation of the language- specific service. SOA-based systems can therefore be independent of development technologies and platforms (such as Java, .NET etc), and can be made to interact between platforms.

SOA can support integration and consolidation of activities within complex enterprise systems, but does not specify or provide a methodology or framework for documenting capabilities or services.

Service Desk's Web Services Interface enables third party applications to interact directly with the Service Desk application in a secure and controlled fashion. The interface is based on the SOAP standard for Web Services. For more information about SOAP Web Services, please see W3C.

SOAP Messages are essentially well formed XML blocks sent using http.

All examples herein are given in Java using the Apache Axis web services library. You can also do your own parsing using XML but that is out of scope for this document. Names throughout this document are exclusive of any suffixes that will be added to names by Web Services Frameworks. The framework will append the word 'Request' to any message sent to the Service Desk Web Services Interface. Similarly, the word 'Response' will be appended to any message name returned to the Customer.

The easiest way to use Web Services is through an API that will wrap the XML parsing and calling for you. Most major Web Services from Google, Amazon and eBay operate this way. Several APIs exist for SOAP but the most prevalent is Axis from the Apache Group. If you are not familiar with the Axis API, please see the Axis User's Guide. The Axis API Reference is also a useful guide if you are interested.

Versions of Axis are available for both Java and C++. Web Services in the Service Desk application conform to the SOAP specification, so there is no reason why you could not use the XML directly from the server and parse it directly. This will require the description document that explains the object types returned and provides other useful information. Use of the Web Services API via direct XML parsing is outside the scope of this document.

While Axis is not required – the examples use Axis as it is one of the most common web services API's in use. Successful customer implementations have also been achieved using the XFire and CXF API's, .NET and Perl.

Session management is equally as important when using web services as it is when using Service Desk via the user interface. It is up to the user (or the client API) to maintain the http headers returned from the authentication message so that state information is retained.

1.11 PHP Web Services

PHP is available as both a command line tool and web programming language and provides excellent services for communicating using standard SOAP protocols. This article describes how to use a PHP command line script to communicate directly with Service Desk.

With the widespread popularity of PHP for web programming, many people have expressed interest in writing scripts to communicate with Service Desk's Web Services rather than using more complex languages like Java.

Many script languages now have plugins for working with Web Services. In this example we are going to discuss PHP. Equally, you can achieve the same thing using languages like Perl.

In order to communicate with Service Desk's web services you should have a thorough knowledge of SOAP and also Service Desk's Web Services API's. Service Desk has a developers guide which details each of the API calls and how to use them. What follows are scripts you can use to handle some of the more difficult parts of communication. Notably session management.

The following code assumes that PHP is installed and running with the PHP SOAP extension. You will note how the cookie is used to manage session state. Feel free to use these functions in your own scripts to facilitate rapid prototyping of your own solutions to communicate directly with Service Desk.

```
<?php
    /*
      LiveTime example PHP command line script for web services
      Requires the PHP SOAP extension to operate
    */

    $hostAddress = "www.host.com";
    $baseProduct = "LiveTime";
    $baseAddress = "/" . $baseProduct . "/WebObjects/" . $baseProduct . ".woa/ws/";
    $baseService = "http://" . $hostAddress . $baseAddress;

    function login($username,$password) {
        global $objClient;
        /*
          Setting "trace" will allow us to view the request that we are making,
after we have made it.
        */
        # Connect to LiveTime Authenticate?wsdl
        # Use only technicians or supervisors
        $results = $objClient->connect($username, $password);

        //This will return an Associative Array
        $success = $results['success'];
```

```
        if ($success=="false") {
            echo $results['message'] . "n";
            exit();
        }
        echo "Login Successfuln";
        createSession();
    }

    function logout() {
        global $objClient;
        global $baseService;

       $objClient->SoapClient($baseService . "Authenticate?wsdl", array('trace' =>
true));

        $results = $objClient->disconnect();
        //This will return an Associative Array
        $success = $results['success'];
        if ($success=="false") {
            echo $results['message'] . "n";
            exit();
        }
        echo "Logout Successfuln";
        exit();
    }

    function createSession() {
        global $objClient;
        #
        # Retrieve the Last Response Header from the LiveTime server

        $responseHeader = $objClient->__getLastResponseHeaders();

        $sessionPos=strpos($responseHeader, "JSESSIONID=");

        if ($sessionPos === false) {
```

```
            echo "No session id was found. Exiting.n";
            exit();
        }
        //cookie will always be 32 bytes
        $cookie = substr($responseHeader,$sessionPos + 11,32);

        #
        # Set the Cookie name for the next request
        $objClient->__setCookie("JSESSIONID", $cookie);
    }

    function findUser($username,$email,$first,$last) {
        global $objClient;
        global $baseService;

        //switch to the customer api
        $objClient->SoapClient($baseService . "Customer?wsdl", array('trace' =>
true));

        echo "Searching for " . $username . $email . $first . $last . "n";
        #
        # Call findCustomer, using an email address and echo out response
        # username, email, first, last
       $results = $objClient->findCustomer($username,$email,$first,$last);
        $success = $results['success'];
        if ($success=="false") {
            echo $results['message'] . "n";
            return;
        }
        #if we have results lets show them
        #returns an array of arrays
        foreach(array_values($results) as $ess) {
            print_r($ess);
        }
    }
    //construct the soap client to create a connection and then login
```

```
    $objClient = new SoapClient($baseService . "Authenticate?wsdl", array('trace'
=> true));

    //if using AD or LDAP you need to enter the Fully Qualified AD/LDAP Domain

    //super@domain.mycompany.com

    login("super","super");

    findUser("","cbrown","","");

    //we are all done

    logout();

?>
```

In the example function above we first login using the connect method of Service Desk's Authenticate wsdl and this subsequently calls the createSession function. To maintain session we grab the JSESSIONID from the response header and then extract the 32 byte cookie. We can now use this cookie in subsequent web service calls to Service Desk. The complete sample code is available for download. Do not forget to logout after completing all the tasks in your code.

Once you have developed your first few scripts you will see that it is possible to do anything you like with the data you obtain from Service Desk. This can be used for coordinating information from multiple data stores or live population of the CMDB from custom systems.

1.12 Java Web Services

Java is the default language for Service Desk. There are many ways to program web services using Java, as there is a rich infrastructure already present in the language. The high transactional rate and scalability of the language makes it perfect for more complex Web Services applications, fault tolerance and communication with other applications.

We have provided a link to the complete source code of a more complex application. Our Web Services are based on Apache Axis 1.x in these examples for the greatest compatibility with existing applications.

In order to communicate with Service Desk's web services you should have a thorough knowledge of SOAP and also Service Desk's Web Services API's. Service Desk has a User guide which details each of the API calls and how to use them. What follows are scripts you can use to handle some of the more difficult parts of communication. Notably session management.

```
package com.livetime.sample;

import com.livetime.ws.cust.Customer_PortType;

import com.livetime.ws.cust.Customer_Service;

import com.livetime.ws.cust.Customer_ServiceLocator;

public class CreateCustomer extends BaseClient {

  /* Customer Service URL */

  public static final String customerServiceURL = "http://10.0.1.11/LiveTime/
WebObjects/LiveTime.woa/ws/Customer";


  /* Constructor - doesn't need to do anything coz this is just a sample */
```

```
  public CreateCustomer() {


  }


  private boolean createCustomer() {
    java.net.URL custURL = null;


    try {
      // Service endpoint
      custURL = new java.net.URL(customerServiceURL);


      // Get handle to the service
      Customer_Service service = new Customer_ServiceLocator();
      Customer_PortType port = service.getCustomer(custURL);


      // Call BaseClient method to populate persistent headers
      populateHeaders((javax.xml.rpc.Stub)port);


     // create properties you want to set for this person - properties you may want
to set here:


      // custom1 (String 255), custom2(String 255), custom3(String 255),
custom4(String 255), custom5(String 255),
      // aliases (comma separated list of email addresses), orgunit (String - case
insensitive match to a LiveTime org unit),
      // phone(String 32), phone2(String 32), phone3(String 32), address(String
128), addressTwo(String 128), postalCode(String 32), city(String 64),
      // country (Integer as String - must be id from getCountries() call),
state(Integer as String - must be id from getStatesForCountry(String countryId)
call)


      java.util.HashMap<String, String> properties = new java.util.HashMap<String,
String>();
      properties.put("phone", "+1 949 777 5800");
      properties.put("phone2", "+61 3 9620 7588");


     // So this is generally reusable without scrubbing database each time, we add
a random number after the username & email
```

```
     // wouldn't do this in the real world obviously - but I'll mention it in case
someone tries :p
      String suffix = Integer.toString((new Double(Math.random() *
1000000)).intValue());


      // username, email, first name, last name, properties (above)
      java.util.HashMap response = port.createCustomer("bob_" + suffix, "bob_" +
suffix + "@mycompany.com", "Bob", "Nelson", properties);


      // Just output stuff from here on
      String successString = response.get("success").toString();
      boolean successful = Boolean.parseBoolean(successString);


      if(!successful) {
        System.out.println("Customer Creation failed: " + response);
      }
      else {
        System.out.println("Customer Creation succeeded: " + response);
      }


      return successful;
    }
    catch(Exception ex) {
      ex.printStackTrace();
      return false;
    }
  }


  // Entry point for execution
  public static void main(String[] args) {
    // create instance of this class
    CreateCustomer createCustomer = new CreateCustomer();
    // try and connect, if successful, create customer and logoug
    if(createCustomer.connect()) {
      // create the customer
      createCustomer.createCustomer();
      // logout
```

```
      createCustomer.disconnect();
    }
  }
}
```

1.13 VB.NET and Service Desk ITSM

This article describes how to use a VB.NET script to communicate directly with Service Desk via Web Services. When using .NET, the address of all the service names, needs to be prefixed with an _, to utilize the newer .NET interfaces.

The service's endpoint address will be the following:

http://HOST_NAME/LiveTime/WebObjects/LiveTime.woa/ws/_SERVICE

As such, the XML Descriptor of the service (WSDL) can be found here:

http://HOST_NAME/LiveTime/WebObjects/LiveTime.woa/ws/_SERVICE?wsdl

The methods for adding web services differs slightly between versions but the general steps are the same. Using Visual Basic Express 2010, to add these into a solution, go to Project > Add Service Reference, Select Advanced and then Add Web Reference. (A common mistake is to omit those last 2 buttons, which are required in the 2010 version). This can be repeated for each service URL required.

The web services commands that rely on authentication, require that the user logs in first using the Connect command and then subsequent commands need to be part of the same session which was instigated by a successful Connect command. This is done by ensuring the JSESSIONID cookie that is set up when the Connect succeeds is then passed back to the server with each command required.

Here is a Windows Form example written in Visual Basic 2010 using .NET 4.0 as the reference basis, although this should work with earlier versions too. Using the initial form that is generated, a button was placed on the form to call that example code. It illustrates logging in, storing the JSESSIONID cookie, adding that cookie to the next command before sending the next command.

```
Imports System.Net

Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        Dim _Uri As New Uri("http://demo.livetime.com/LiveTimeOpen")   'This should
match the address of the web services references added to the solution

        Dim _InitialCookieContainer As New CookieContainer

        Dim _CookieContainer As New CookieContainer

        Dim _Cookie As New Cookie

        Dim _LoginSuccess As Boolean = False

        'Login

       Dim LTAuth As New LTAuthenticate._Authenticate   'LTAuthenticate is the name
given to the Authenticate web service of LiveTime in the solution

        Dim _auth_response() As LTAuthenticate.WsNameValuePair
```

```
        LTAuth.CookieContainer = _InitialCookieContainer

        _auth_response = LTAuth.connect("super", "super")   'Login details

        For Each _response In _auth_response

            If _response.key = "success" And _response.value = "true" Then

                _LoginSuccess = True

            End If

        Next

        'If successfully logged in, store the JSESSIONID for subsequent use.

        'Alteranatively, all the initial cookies can be referenced without
extracting just the required JSESSIONID one but this example shows what is required
in LiveTime.

        If _LoginSuccess Then

            _Cookie = _InitialCookieContainer.GetCookies(_Uri).Item("JSESSIONID")

            _CookieContainer.Add(_Cookie)  'Setup a separate CookieContainer with
just the required JSESSIONID cookie for subsequent requests

        Else

            MessageBox.Show("Login Failed")

            Application.Exit()

        End If

        'Example to find customers

        Dim LTCust As New LTCustomer._Customer   'LTCustomer is the name given to
the Customer web service of LiveTime in the solution

        LTCust.CookieContainer = _CookieContainer   'The Customer commands are to
use the CookieContainer created above - Alternatively, in this case, it could also
use the initial one here, as mentioned above.

        Dim _cust_response() As LTCustomer.WsNameValuePair

        _cust_response = LTCust.findCustomer("", "", "")   'Calling findCustomer
with email, first name, last name

        'Parse the result to a message box, displaying 4 at a time

        Dim _pair As LTCustomer.WsNameValuePair

        Dim _total_response As String = ""

        Dim _count As Integer

        _count = 0

        For Each _response In _cust_response

            If TypeOf (_response.value) Is String Then

                If _response.key = "message" Then

                    _total_response = _response.value

                End If

            ElseIf Not (_response.key = "success") Then
```

```
                _total_response = _total_response + _response.key + " : " +
vbNewLine
                _count = _count + 1
                For Each _pair In _response.value
                    _total_response = _total_response + _pair.key.ToString + " : "
+ _pair.value.ToString + vbNewLine
                Next
                _total_response = _total_response + vbNewLine
            End If
            If _count >= 4 Then
                MsgBox(_total_response)
                _total_response = ""
                _count = 0
            End If
        Next
        If _count <> 0 Then
            MsgBox(_total_response)
        End If
    End Sub
End Class
```

If using another language (eg. C#) or type of .net application, the same principles apply so the Visual Basic Windows Form example can be used as a guide.

1.14 Nagios CMDB Closed Loop Integration

1.14.1 Introduction

Nagios is a popular event management application. The scriptable commands and plug-ins, is very customizable for performing any action when a monitor alert is triggered. This document will demonstrate how to communicate directly with Service Desk's CMDB and update CI's in real time with bi-directional feeds back to Nagios and Service Desk.

When a check fails and the monitor alerts, one common result can be that an incident is created whilst the failing service is investigated. With the event-handler capability of Nagios and Service Desk's inbound web services, this is simple to setup.

Additionally, with Service Desk's outbound web services and run book automation the event-handler can be paused and re-instated after the incident has been dealt with. This makes it easy to have the Nagios alert create just one request and then only create new ones once the issue with the service has been resolved. This is especially useful for services that are intermittently working/failing and alerting on the Nagios monitors.

Furthermore, if the CMDB in Service Desk is configured correctly, it can be possible to identify the item in question and take the item offline in Service Desk, restoring the item after the incident has been resolved.

1.14.2 Operational Workflow

1. Service fails and alerts in Nagios.
2. The Nagios event handler calls a PHP script.
3. The PHP script parses the host and service names from Nagios and utilizing Service Desk's inbound web services, it determines the item in the CMDB in question. If the Item for the service is not found that is related to the host, then it falls back then it falls back to the default item configured in the script.
4.
5. An incident is raised against the item established in step 3.
6. If an exact item match is found in step 3, the item's status is taken offline.
7. If the incident is successfully created, to prevent further requests being created, the event handler is disabled for the alert in Nagios via a web request with the host and service parsed in step 3.
8. The Nagios check that triggered this is recorded as the first note on the incident.
9. Once the issue is resolved, the incident is moved into a status which has an outbound web services script attached to it.
10. Outbound script reads the first note on the incident to establish the Nagios check involved and re-enables the event handler on Nagios for this, via a web request.
11. It also updates the item on incident back to it's available status, if an item was identified in step 3.

1.14.3 Setup

For the purposes of this document, the Nagios Virtual Machine was used. This is available for download from the Nagios website. If Nagios is already setup and running, then skip to the last step of these instructions as ensuring php-soap is installed is the most important step here.

Once running, the admin pages are at: http://host/nagiosxi/

The user pages are at: http://host/nagios/

To avoid having to rely on accessing the console output of the virtual machine, the firewall needs opening up to allow local users access. To do this, for all ports, on a 192.168.1.0/24 based network (for example), one would login as root using the default password of nagiosxi and run:

```
iptables -I INPUT -p tcp -s 192.168.1.0/24 -j ACCEPT

iptables-save

service iptables save
```

Some checks then need to be added via the admin pages. For the example in this document, we add an FTP check to a Windows server named Windows 101.

If the auto discovery wizard is to be used to quickly get some checks added, where you pick which machines you want to check after it completes, traceroute needs to be installed first.

To do this login as root and run:

```
yum install traceroute
```

To get files to/from the virtual machine, one method my be to use scp, and to do that, you would need to first run:

```
yum install openssh-clients
```

Finally, PHP is already installed on the virtual machine but for web services commands to run, php-soap also needs to be installed. This is done by running:

```
yum install php-soap
```

1.14.4 Creating the Request in Service Desk

A web services script is created to do this, and this is copied to the Nagios server. There is a scripts folder already setup on Nagios so in our example we copied the script as:

```
/usr/local/nagiosxi/scripts/lt_nagios_create_request.php
```

In Nagios go to the admin, **Configure** > **Core Configuration Manager**. Select **Commands drop down** > **Commands** > **Add new**.

Enter these values:

1. **Command**: lt-create-request
2. **Command Line**: /usr/local/nagiosxi/scripts/lt_nagios_create_request.php $SERVICESTATE$ $SERVICESTATETYPE$ $SERVICEATTEMPT$ $HOSTNAME$ $SERVICEDESC$
3. **Command Type**: Misc
4. **Active**: Ensure this check box is selected.

Click **Save**.

To complete this addition, click **Apply Configuration**.

Next, Under Monitoring, Click **Services**.

Find the service that needs to create a request on alert (in our example, the Config Name is ‘Windows 101′ and the Service Name is ‘FTP’) and click tools icon to edit it.

Under the Check Settings tab, use the drop down for Event handler to select lt-create-request and set the enabled option to on.

Click to save those settings and then click **Apply Configuration** to complete this modification.

1.14.5 Configuring the Service Desk CMDB

For the correct item to be used, there are a number of ways to do this, but in our example PHP script, we set it up so the host is an item in Service Desk, and the Nagios Service being tested is also an item in Service Desk.

For instance, you might have a server numbered ‘Windows 101′ as the host with and FTP service running on it, which for this we have assigned an item numbered ‘FTP101′, which is of the type ‘FTP’. The items are then setup in a relationship so that the ‘FTP101′ is installed on ‘Windows 101′.

The number of the actual service item doesn’t matter but in our example the number of the host item does, if items are not to be assigned to the default item.

The example script first attempts to locate the item numbered the same as the host, ‘Windows 101′ and then looks for a related item of the type matching the service being checked, ‘FTP’. It finds the item ‘FTP101′ so it picks that item for the new incident.

If it didn’t find an FTP type of item related to the host, it would just raise it would just raise it against the default item.

The script will also update the item status to an Offline one but only if it finds an exact match to the service. If not, it will not update it. This is purely in the example script so obviously can be configured as required.

Also, the example script is not strict on the relationship type, for the purposes of demonstrating this. It could be further amended to only look at services installed on the host, as opposed to those also used by the host, if other relationships existed in the CMDB in question. This could readily be added to the script.

1.14.6 Re-activating the Event Handler

Service Desk’s outbound web services can be used to call a script to reactivate the event handler in Service Desk once the request has reached a status where the service issues have been resolved.

Additionally, the script can call back to Service Desk’s inbound web services to update the item’s status as online, if an exact item was identified and placed into an offline state in the original script that created the request.

As a Java application, the scripts behind Service Desk’s outbound web services need to be in Java.

For the best performance, the script that performs these actions should ideally be coded in Java. However, for the purposes of this demonstration, it is also possible to code enough java to call another command, which can then be coded in any language. In our case, we are going to call a PHP script as we can then reuse some of the code used in the initial script too.

The script sends, as command line arguments, the status change involved (ie. if the status was Entered or Exited), the request number and the item number on the request. The PHP command then reads these and performs the required actions.

To install the provided code for this, jump to step 3 below and copy the included jar file over to the location specified. However, if making changes to the script, the following needs to be done:

1. To amend the code, copy this file to the folder with the java file in: {LiveTime Install Folder}/LiveTime.woa/Contents/Resources/Java/livetime-listen.jar
2. After all required changes have been made, the following commands need to be entered:

   javac lt_nagios_resolved.java

   jar cf lt_nagios_resolved.jar lt_nagios_resolved.class -classpath livetime-listen.jar
3. The resulting lt_nagios_resolved.jar file then needs to be coped to the following location on the Service Desk server: {LiveTime Install Folder}/LiveTime/WEB-INF/lib/

After this, the Service Desk application server, needs to be restarted for this to be picked up.

By default, the java code is calling ‘php /root/lt_nagios_resolved.php’. As such, php and php-soap need to be installed on the Service Desk server, along with copying lt_nagios_resolved.php to the /root/ folder.

Note that the PHP script is set to change the status of the service to ‘Available’ in line with the online status provided by default for items of the Service category in Service Desk. This can be modified according to what is in use.

In fact, the script could be further developed to look for ‘Available’ and if that wasn’t present, try ‘Online’, etc, if multiple item category types were used. Obviously that is up to the user to add such coding.

Finally, to call this java code, the workflow used for dealing with the alert requires a status that is linked to the class in the jar file created above. To do this, ‘Outbound Web Services’ needs to be enabled under **Setup** > **Privileges** > **System** and the status in question is then updated so that it’s Listener Class is: lt_nagios_resolved.

1.15 ZSD Extensions

ZENworks Service Desk (ZSD) has the ability to add customized extensions to request workflow state transitions and item lifecycle state transitions. In order to implement custom functionality on these state transitions, there are several steps a user must follow:

1.15.1 Building the Extension

Two java interfaces are provided, in a standalone jar file ‘livetime-listen.jar’, which can be found in:

```
{ZSD Installation Folder}/LiveTime.woa/Contents/Resources/Java/
```

This jar file contains two interfaces, each of which has two methods:

```
WorkflowListener
public Map<String, String> stateEntered(Map<String, Object>argsMap) throws Exception
public Map<String, String> stateExited(Map<String, Object>argsMap) throws Exception
LifecycleListener
public Map<String, String> stateEntered(Map<String, Object>argsMap) throws Exception
public Map<String, String> stateExited(Map<String, Object>argsMap) throws Exception
```

Naturally the WorkflowListener is used for integrating with the Request Workflow, whilst the LifecycleListener is used for integrating with the Item Lifecycle.

The task for the developer is to create a java class that implements the appropriate interface to achieve the integration objective. The two methods exist to provide flexibility to the developer implementing the integration, by allowing them to perform tasks based on a state being ‘exited’ and then a state being ‘entered’.

The implementing class is required to return a Map for all methods. Non-implemented interfaces can return null, and this will be treated as a no-op internally. Methods that provide functionality should return a map which will be parsed for two parameters:

- ‘success’ : ‘true’ or ‘false’
- ‘message’ : Description to be stored against the history of the request or item

Each method is passed a Map of parameters that relate to the request or item being updated. ZSDZSD will pass a String for all values, the method takes for future extensions that may require the use of Objects. Implementations should check the object is a String prior to casting to future-proof the implementation.

1.15.2 WorkflowListener Arguments

The parameter Map passed in to the WorkflowListener interface methods consists of:

Parameter	Description
triggerStatusId	The ID of the status that has triggered the listener event: ◆ statusExited: originalStatus ◆ statusEntered: newStatus
triggerStatusName	The name of the status that triggered the listener event
requestId	The ID of the request being updated ◆ 1000 = Incident ◆ 2000 = Problem ◆ 3000 = Change Request ◆ 7000 = Service Request ◆ 9000 = Deployment Task
requestType	The type of request being updated ◆ 1000 = Incident ◆ 2000 = Problem ◆ 3000 = Change Request ◆ 7000 = Service Request ◆ 9000 = Deployment Task
statusId	The current state of the request: ◆ statusExited: newStatus ◆ statusEntered: same as triggerStatus
statusName	The name of the state defined by statusId above
classificationId	The ID of the classification assigned to the request
customerId	The ID of the customer assigned to the request
customerFirstName	The first name of the customer assigned to the request
customerLastName	The last name of the customer assigned to the request
customerEmail	The email address of the customer assigned to the request
orgUnitId	The ID of the Org Unit associated with the request
orgUnitName	The name of the Org Unit associated with the request

Parameter	Description
itemNumber	The item number of the CI associated with the request

As this is a first iteration of the call-out interface, these parameters have been deemed sufficient to allow a developer to perform non-trivial tasks like update an external system that may require request updates.

1.15.3 LifecycleListener Arguments

The parameter Map passed in to the LifecycleListener interface methods consists of:

Parameter	Description
triggerStatusId	The ID of the status that has triggered the listener event: ◆ statusExited: originalStatus ◆ statusEntered: newStatus
triggerStatusName	The name of the status that triggered the listener event
itemNumber	The Item Number assigned to the CI
itemStatusId	The current state of the request: ◆ statusExited: newStatus ◆ statusEntered: same as triggerStatus
itemStatusName	The name of the state defined by statusId above
itemtypeId	The Item Type ID
itemtypeName	The name of the Item Type the Item is an instance of
categoryId	The Category ID
categoryName	The name of the Category the Item Type belongs to

These are the initial parameters deemed appropriate to allow a developer to perform non-trivial tasks like (for example) to feed state changes into a monitoring tool to reset an alert trigger.

1.15.4 Implementing a Listener

This requires some Java knowledge, to either define the entire functionality, using these entry points, or to create a JNI wrapper to page out to code written in an alternate language, although this does have platform implications. A JNI (Java Native Interface) implementation is outside the scope of this document, and the following sample focuses on a Java Implementation.

The user needs to create a class, for example ‘MyWorkflowListener’ which implements the WorkflowListener Interface, or MyLifecycleListener, which implements the LifecycleListener interface. These methods then need to be made to perform some work. A non-trivial example would be one where the listener calls a web service to an external system. Consider the following usage scenario.

A company has (for whatever reason) two ZSD instances, and they are needing to, on occasion feed updates to the secondary system on request state changes. ZSD has an inbound web services interface, and now, an outgoing interface for communicating changes to third parties.

The SOAP WSDL's can be used to generate Java classes to make calls into the secondary system, which can now be called from the outgoing interface. Generating the SOAP equivalent java classes, and calling them from a WorkflowListener might yield a listener class that looks something like the following.

```
package com.livetime.sample;
import com.livetime.ws.listen.WorkflowListener;
import java.util.Map;
public class WorkflowListenerImpl implements WorkflowListener
{
public WorkflowListenerImpl() {}
public Map<String, String> statusEntered(Map<String, Object>argsMap) throws Exception {BaseRequest
baseRequest = new BaseRequest();
String requestId = (String)argsMap.get("requestId");
String statusName = (String)argsMap.get("triggerStatusName");
Map temp = new HashMap();
temp.put("subject", "Request Created via Outbound WebServices Call");
temp.put("description", "Request #" + requestId + " has entered status " + statusName);
return baseRequest.createRequest(temp);
}
public Map<String, String> statusExited(Map<String, String>argsMap) throws Exception
{
BaseRequest baseRequest = new BaseRequest();
String requestId = (String)argsMap.get("requestId");
String statusName = (String)argsMap.get("triggerStatusName");
Map temp = new HashMap();
temp.put("subject", "Request Created via Outbound WebServices Call");
temp.put("description", "Request #" + requestId + " has exited status " + statusName);
return baseRequest.createRequest(temp);
}
}
```

With the createRequest method in the class BaseRequest looking like this:

```
public Map<String, String> createRequest(Map<String, String>
properties)
{
// try and connect, if successful, create request and logout
 if(connect()) {
java.net.URL requestURL = null;
HashMap response = new HashMap();
try {
// Service endpoint
requestURL = new
java.net.URL(props.getRequestServiceURL());
// Get handle to the service
Request_Service service = new
Request_ServiceLocator();
Request_PortType port = service.getRequest(requestURL);
// Call BaseClient method to populate persistent
headers populateHeaders((javax.xml.rpc.Stub)port);
String subject = (String)properties.get("subject");
String description = (String)properties.get("description");
response = port.createIncident(props.getTargetItemNumber(),
props.getTargetClassificationId(), subject,
description, new HashMap());
}
catch(Exception ex)
{
return buildErrorMessage("An error occurred whilst creating");
}
if(disconnect())
{
return response;
}
else {
return buildErrorMessage("An error occurred whilst disconnecting");
}
}
else {
return buildErrorMessage("An error occurred whilst connecting");
}
}
```

In this example, the listener simply creates a new request in the second system, stating the nature of the change, but this highlights some of the possibilities of outbound functionality.

NOTE: This example has been heavily truncated to illustrate the key functionality.

1.15.5 Making the Extension Available to ZSD

This is a rather simple process for users with an install on their own infrastructure.

In order for the class to be accessible to ZSD, the compiled code needs to be on the ZSD classpath. In this case, this means the compiled jar, along with any associated components, need to be copied into the lib folder of the ZSD installation ({ZSD Path}/WEB-INF/lib), and the ZSD instance needs to be restarted, so these resources are picked up by the classloader.

1.15.6 Configuring ZSD to Use the Extension

At this point a new class (or classes) exist and have been loaded, now ZSD simply needs to be configured to use them. This is a two part process. Firstly the option needs to be enabled by a system administrator to enable outbound web services. This option can be found in the Administrator portal, under **Setup** > **Privileges** > **System** (Outbound Web Services).

Once set to 'On' and saved, a new field appears in both the Workflow State and Lifecycle State Editors respectively. The 'Listener Class' can now be populated per workflow (or lifecycle) state, allowing each state to call the same, or different implementations as necessary. This allows different workflows to behave per the designers' requirements.

1.16 ZSD Store Extensions

ZENworks Service Desk (ZSD) has the ability to add the customized store extensions for the auto assignment of store item. In order to implement custom functionality on the assignment state follow:

1.16.1 Building the Extension

Java interface provided in a standalone jar file `livetime-listen.jar`, and it is available at:

`{ZSD Installation Folder}/LiveTime.woa/Contents/Resources/Java/`

The `livetime-listen.jar` file contains the `ExternalStoreExtension` interface, with the following method:

`public Map<String, String> statusEntered(Map<String, Object> argsMap)` throws `Exception`;

The developer task is to create a java class that implements above interface to achieve the integration objective.

The implementing class is required to return a Map. Method that provide functionality should return a map which will be parsed for the following parameters:

- `success`: '`true`' or '`false`'
- `message`: Description to be stored against the history and note of the store request.

The method is passed a Map of parameters that relate to the store request being updated. ZSD will pass a string for all the values, this method takes for future extensions that might require the use of objects. Implementations should check whether the object is a String prior to casting to future-proof the implementation.

1.16.2 ExternalStoreExtension Arguments

The parameter Map passed in to the `ExternalStoreExtension` interface method consists:

Parameter	Description
`triggerStatusId`	ID of the status that has triggered the extension: ◆ `statusEntered`: `newStatus`
`triggerStatusName`	Name of the status that triggered the extension.
`requestId`	ID of the request being updated.
`requestType`	Type of the request being updated. ◆ 7000 = Service Request
`statusId`	Current state of the request same as assignment state.
`statusName`	Name of the state defined by statusId above.
`classificationId`	ID of the classification assigned to the request.
`customerId`	ID of the customer assigned to the request.
`customerFirstName`	The first name of the customer assigned to the request.
`customerLastName`	The last name of the customer assigned to the request.
`customerEmail`	Email address of the customer assigned to the request.
`orgUnitId`	ID of the Org Unit associated with the request.
`orgUnitName`	Name of the Org Unit associated with the request.
`itemNumber`	Item number of the CI associated with the request.

As this is the first iteration of the call-out interface, these parameters have been deemed sufficient to allow a developer to perform tasks such as update an external system that might require request updates.

1.16.3 Implementing the Store Extension

Requires Java knowledge, to either define the entire functionality, using these entry points, or to create a Java Native Interface (JNI) wrapper to page out to code written in an alternate language, although this does have platform implications. A JNI implementation is outside the scope of this document, and the following sample focuses on a Java Implementation.

The user needs to create a class, for example '`MyStoreExtensionImpl`' which implements the `ExternalStoreExtension` interface. Implemented method then needs to be made to perform some work. For example, the extension calls a web service to an external system. Consider the following usage scenario:

```
package com.microfocus;

import java.util.HashMap;
import java.util.Map;
import com.novell.store.extension.external.ExternalStoreExtension;

public class BusinessCardExtensionImpl implements ExternalStoreExtension
{
       /**
     * This method will be called at the entry of the state, i.e if workflow has state hierarchy like
state1-> Assignment State-> State2 then this will be called on transition of state1 to Assignment
State
     * @param argsMap
     * @return a map. This map must contain 2 entries.
     *         key=message value=A user friendly message summarizing the result of action performed by
this extension.
     *          key=success value=true|false in String. this denotes the status of the extension
process.
     * @throws Exception
     */
       @Override
     public Map<String, String> statusEntered(Map<String, Object> arg0)
      throws Exception {
    boolean success = doSomething(); // Your actual business logic goes here.
    Map<String, String> response = new HashMap<>(); //This message will be added as part of the Note
and Audit Trail. E.g

    if(success)
             response.put("message", "Order for Business card has been successfully placed.");
    else
             response.put("message", "Order for Business card has been failed.");

           response.put("success", String.valueOf(success));

           return response;
  }

  private boolean doSomething() {
    //
    //Do Something. Invoke external services handling order for business cards.
    //
    boolean success = true;
    return success;
    }
}
```

1.16.4 Making the Store Extension Available to ZSD

In order for the class to be accessible to ZSD, the compiled code needs to be on the ZSD classpath. In this case, the compiled jar, along with any associated components, need to be copied into the lib folder of the ZSD installation (`{ZSD Path}/WEB-INF/lib`), and the ZSD instance needs to be restarted, so these resources are picked up by the classloader.

1.16.5 Configuring ZSD to Use the Extension

At this point a new class or classes exist and have been loaded, now ZSD needs to be configured to use them.

- Enable the Store feature. For information, see *Enabling Store*.
- Create the Store Extension. For information, see *Creating a Store Extension*.
- Assign the extension to applicable Item Category or Item Type.