Configuring Forms for a Service

This chapter contains the following topics:

Configuring Forms for a Service

Introduction

A service form is used by an end-user to define their service request. Using Service Designer module, you can configure the service forms, and create data retrieval rules to display specific choices when ordering services. Service Form design uses these components:

- **Dictionaries** are group of fields to enter for a service request. There are four types of dictionaries that can be added in Prime Service Catalog. To know more about these dictionaries, see Configuring Service Form Fields Using Dictionaries, on page 1.
- **Standards** that aid in designing the active form components. To know more about standards, see Defining Standards in Service Design, on page 18.
- **Active Form Components** help in defining service form's appearance and behavior based on user-initiated events. To know more about the Active Form Components, see Configuring Service Form Appearance and Behavior Using Active Form Components, on page 22. You can also create rules to alter the behavior and appearance of a service form based on specified conditions. For more information, see Configuring Dynamic Form Behaviors Using Form Rules, on page 33.

Configuring Service Form Fields Using Dictionaries

The basic building block of a service form is a dictionary, a group of individual data elements (fields) that allow users and service performers to enter and view data required to fulfill the service request. The dictionary should incorporate all fields that may be required in these request.

This principle (of designing a dictionary and form component for reuse) is particularly important for Service Item-Based Dictionary, on page 3, since services may be available for the entire lifecycle of the service item, not just the initial request to provide a service (or item) to the customer. You may want to design the

dictionary to support the tasks that can be included in a service request; create the service item; modify the service item; or remove the service item from its current owner.

Types of Dictionaries in Prime Service Catalog

The two main categories of dictionaries, Internal and External, refer to the way data in the dictionary is stored.

- Internal dictionaries represent data structures that are managed by, and within, Service Catalog. Internal dictionaries are generally preferable, since they minimize database administration required and include additional functionality not available in external dictionaries. Types of Internal dictionaries are:
  - Internal Free-Form Dictionaries, on page 2
  - Internal Person-Based Dictionaries, on page 2
  - Service Item-Based Dictionary, on page 3
  - Automatically Included Reserved Dictionaries, on page 7
  - Automatically Included Integration Dictionaries, on page 9
- External dictionaries, use existing or new data tables outside the Service Catalog requisition. For more information, see External Dictionaries, on page 6.

Internal Free-Form Dictionaries

A free-form dictionary means that service designers are free to specify the fields in the dictionary, the order in which they occur, and the data types assigned to each. To configure the fields in the free-form dictionary, see Table 2: Fields to Configure Internal Dictionary.

Internal Person-Based Dictionaries

Cisco Prime Service Catalog maintains a repository of all persons in the user organization who may need to access Prime Service Catalog applications. Personnel information in Service Catalog is viewable via Organization Designer. The personnel information in this repository is generally populated via directory integration. Any time a user logs in to Service Catalog, or has a service ordered on his/her behalf, his/her personnel information in the repository can be refreshed with their information from an enterprise-wide directory. Service forms typically need to refer to personnel information. For example, a service request always has a customer—the recipient of the service; and an initiator or requestor—the person who sits at the keyboard and requests the service. In most cases, the customer and the initiator are the same person—an employee requests a service for him/herself. Alternatively, an administrator or other authorized employee may initiate a service request on behalf of another person, who then becomes the customer for the service. To configure the fields in this type of dictionary, see Table 2: Fields to Configure Internal Dictionary table.

When to use Person-Based Dictionaries?

If you are only referring to the request's as a customer or an initiator, you do not have to create a person-based dictionary. Instead, use the Customer_Information and Initiator_Information dictionaries that are part of the Reserved dictionaries in Prime Service Catalog.You have to create person-based dictionaries when requesters frequently need to designate another person who may be involved in the fulfillment of the request or needed as a reference.

Use case for Person-Based Dictionary

A frequent use case for person-based dictionaries is when an initiator designate one or more approvers for a request, if the approvers cannot be inferred from the supervisory structure available from the directory integration. In this scenario, the requester selects the approver by searching through a drop-down list of available personnel and the approver's contact information is then included in the service form data for easy reference.

About Person Search Option

When a person-based dictionary is used on a service form, you can choose a single person specifying the search criteria in the “Person Search” dialog box. Once a person is chosen, fields used in the dictionary are automatically filled in with the current values of the corresponding fields in the chosen person’s profile. The name appears in FirstName LastName format.

The “Select_Person” attribute provides the capability to search for people, either within the repository or, if directory integration has been enabled for service forms, in an external directory. This attribute has a data type of “Person,” not text. This data type governs the appearance of the attribute when it is used in a form. For example, the “Name” field on the service form below is defined as the “Select_Person” attribute. In any new person-based dictionary, you will see “Show in Grid” is checked by default for the “Select_Person” since it also checked for use Use by default. “Show in Grid” cannot be unchecked for the Select-Person, just as you cannot uncheck Use. Select the other dictionary attributes to be displayed on the service form by clicking the Use checkbox.

Figure 1: Select Person Attribute

Service Item-Based Dictionary

A Service Item-Based Dictionary is a dictionary whose structure (fields) are based on a service item defined in the Service Item Manager module. This dictionary also holds data about that service item collected during a service request. To see type of data that can be configured for a service item, see section on Defining Service Items for a Service.

Fields in that dictionary provide the data stored on the service item and can optionally serve to update the history of the service item and its subscriptions. For more information, see Service Item Subscription and History Fields.

Adding Service Item-Based Dictionary

Using this procedure create a service item-based dictionary based on attributes defined while creating service items,

Before You Begin

Configure Service Item, based on which the dictionary will be defined.

Step 1 Choose **Service Designer** > **Dictionaries**

Step 2 Choose **New** > **New Dictionary** to display the New Dictionary page.

Step 3 In the Add New Internal Dictionary section, in the **Service Item** field, enter one or more characters of the name of the Service Item you want, or enter "*" to search for all Service Items. Click on the Service Item you want.
The values for **Category**, **Service Item Group**, and **Service Item Type** fields are set to the group and service item type you selected. As in previous releases, Service Item Family is not used by Service Catalog and any value may be supplied. This value is available for query and grouping in the Service Catalog data mart. The other fields that can be included in a service item-based dictionary are:

- Fields corresponding to attributes defined in the service item itself. For more information, see Service Item Fields.
- Fields about the service item subscription (history) and delivery history that are available for use in conjunction with the dictionary. These fields provide additional information regarding the service item's current usage and subscription history. For more information, see Service Item Subscription and History Fields.
- User-defined fields. For more information, see User-Defined Fields.

Step 4 Click Save Dictionary.

Note If you want to add a new attribute to the corresponding service item after creating the dictionary, you must manually add the new field. This is because Service Catalog does *not* automatically add a new field corresponding to that attribute to the dictionary. As long as you use the same field name as the attribute name (the Data Name, not the Display Name), Service Catalog will correctly synchronize fields in the dictionary with the attributes in the service item. Similarly, if you create a SIBD and then *delete* an attribute from the corresponding service item, you must manually remove the dictionary field corresponding to the deleted attribute. If you do not, you will end up with a field on the form that has no corresponding data for the service item.

Service Item Fields

Attributes specified in a user-defined service item are, by default, included in dictionaries based on that item. To add additional fields to the dictionary, select the check box on the left of the attribute name under Dictionary Attributes. Any other fields except the **Name** field are optional. The Name field has a reserved data type as "Service Item Identifier". All data types are inherited from the service item's definition. Select the **Encrypt** check box to encrypt sensitive data such as passwords, beneficiary names, account numbers, and so on.

Note Use String (max) field type for service item dictionaries that will be encrypted to ensure that the encrypted string does not exceed the character limit.

The String (max) limit is configurable to 6000 in cnfparam table by the system administrator. However, ensure that the new string limit value does not create performance issues when configured in the service item dictionary. For more information on encrypting sensitive data, see Table 3: Dictionary Attributes table.

Service Item Subscription and History Fields

The next set of fields that appear on the Dictionary page correspond to data maintained for the service item subscription and history. The requisition's subscription and history data is recorded in the Service Item History when the Create or Update Service Item task for the service item is executed. This option for an end user is available in the **My Stuff** page, and for an administrator in the **Service Item Manager > Manage Service Items > History** tab for each item.

Service Item Subscription fields are summarized in the table below. For the pre configured subscription fields, choose the fields you want to include in the dictionary by checking the check box to the left of the field name. The fields listed below are automatically included in the service item's history and subscription data, with appropriate values filled in, even if you do not include them in the dictionary. The reason for including them in the dictionary is to supplement or override the default behavior that Service Catalog uses to provide values to these fields.

Table 1: Service Item Fields

Field Name	Description
CustomerID	ID of the customer for the request/service item.
RequisitionID	The ID of the requisition (shopping cart), which includes the service request that created the service item subscription.
RequisitionEntryID	The ID of the service request that created the service item subscription.
OrganizationalUnitID	The Home OU of the customer to whom the service item is assigned.
AssignedDate	The date and time the service item was assigned to the customer, that is, the date the "Service Item - Create" task was executed.
SubmittedDate	The date and time the request, which included the "Create Service Item" task, was submitted.
ErrorCode	Not used.
ErrorDescription	A textual description of the error code received if an attempt to create or assign the service item failed.
Account ID	By default, the Tenant Account of the Home OU for the customer to whom the service item is assigned.

Note If CustomerID and OrganizationalUnitID are not included in the service item-based dictionary, Service Catalog uses its default logic to supply a value for each field. If a field is included in the dictionary, the value of the form field at the time the service item task is executed is used.

- If CustomerID or OrganizationalUnitID are included in the dictionary, the service designer is responsible for writing a rule or other mechanism to provide a value to the field.

Some sample scenarios for overriding the default behavior of Service Catalog include:

- Create the service item (SI), but with no owner. In other words, no one has a subscription to it. For example, a new laptop has arrived, but it has not yet been assigned to anyone. This would be an alternative to manually creating the service item via the **Manage Service Items** tab or importing the service item from an external source. In this scenario, both the CustomerID and OrganizationalUnitID would be included on the service form (probably hidden by a rule in the Ordering moment), but no value supplied. Add AccountID every time when both CustomerID and OrganizationalUnitID are referenced.
- Allow the request initiator to explicitly choose the customer for the service item (via a person-based dictionary). Copy the customer's information to the CustomerID and OrganizationalUnitID fields included in the SIBD, overriding the default customer (the initiator). Add AccountID every time when both CustomerID and OrganizationalUnitID are referenced. This scenario is explained in more detail in the XREF section.
- Create the SI with no specified owner (a person corresponding to the CustomerID), but with an owning OU. For example, a project team needs a server, but the server is the responsibility of the team, not an individual person. This scenario is similar to the one above, but only the CustomerID needs to be in the dictionary, its contents blanked out by a rule in the Ordering moment. The OrganizationalUnitID can be the OU ID of the initiator or customer, or you could choose an OU/customer via a Person Search function.
- Unassign the service item from someone. For example, when an employee's equipment is temporarily unassigned after he or she leave the company or a project.
- Run a data retrieval rule to display information about when the SI was created (for example, which Requisition and Requisition Entry IDs were on the previous order).

The error description field provides feedback in case a service item task fails. This is a useful for debugging, by giving clues on possible conditional rules that might be required to validate data entry.

Possible reasons for failure to create a service item include leaving the item's Name blank, or attempting to create a service item of the same type with the name of an item that already exists. If such a task fails, the service item is not created, and any changes to the requisition are rolled back. Similarly, a task to update or delete a service item might fail if the referenced item does not exist.

User-Defined Fields

Service item designers may add fields to any service item-based dictionary. Such fields might not be appropriate for the service item itself, but are a critical part of service requests relating to the service item. For example, you may want to include comments on why the customer is requesting the item, or initial or recurring monthly costs associated with item usage. Field names must not match the names of fields from the Service Item History/Subscription available for use, even if those fields are not included in this dictionary.

Such a field just becomes data that lives on the form and in the dictionary data for the form; it will never be recorded as part of the service item and is not viewable via the Service Item History and Subscription queries. However, like any dictionary field, it is viewable in the completed request and reportable via the Advanced Reporting module if the dictionary or a service that includes the dictionary is made reportable.

External Dictionaries

External dictionaries are added and maintained in Service Designer > Dictionaries. Using an external dictionary in read/write mode can result in corruption to the database that you are accessing from the system if you have not set up safe data transfer methods. Read the following tips:

- The Requisition Entry ID generated by the system needs a column and primary key of its own in the external database table. Otherwise, the system can overwrite other data in the table.

- We recommend that your site create an intermediate table with a key column to which the system writes new entries. This table can then work with your existing table.
- Before adding an external dictionary, consult with the external table's DBA or owner to ensure that the system uses the correct columns for the Requisition Entry ID and the primary key.
- The system does not automatically synchronize changes that you make to external database tables with its own external dictionaries that reference those tables. Changes to column names or sizes, for example, will not flow through to external dictionaries, and services ordered that depend on those dictionaries will fail. Once you have created an external dictionary in the system, each time you change the structure of the external database table, you must save the definition page of the external dictionary in the Dictionaries component of Service Designer. When you click Add This Dictionary on this page, the system will synchronize the table structure, and display the new information in the section that describes the external database columns.

Automatically Included Reserved Dictionaries

Every Service Catalog instance automatically includes two dictionaries, provided as seed data for customer and initiator information. These dictionaries can be found in the "Reserved" dictionary group in the Dictionaries component of Service Designer.

There is also a "Reserved" form group in Active Form Components, which contains the Customer-Initiator form. This form includes the Customer_Information and Initiator_Information dictionaries.

Features of Customer_ and Initiator_Information dictionaries:

- Supplement the standard behavior of automatically recording the customer and initiator for all requests.
- All information on the customer and initiator is available throughout the requisition life cycle via Business Engine namespaces, and is displayed as part of the Requisition Summary page for the request in My Services.
- These dictionaries are included in active form components and reusable for all services for the following reasons:
    - These fields reflect the values that were retrieved from the person's profile.
    - The person ordering the service has no opportunity to correct information that may be out of date or to supply additional information needed to fulfill the current request.
    - In addition, for reporting and governance reasons, it may be required to track customer and initiator information as they were when the request was submitted, not reflecting any subsequent changes.
- The dictionaries and group names for reserved dictionaries cannot be modified.

- The reserved dictionaries are included on a reserved form, the Customer-Initiator form. Form contents and appearance can be defined and its behavior manipulated by any form rules or ISF available.

Figure 2: Requisition Summary Page in My Services

The dictionary and group names of the Initiator and Customer Information dictionaries cannot be changed. Other general properties of the dictionaries are editable. The reserved dictionaries are included on a reserved form, the Customer-Initiator form. Form contents and appearance can be defined and its behavior manipulated by any form rules or Interactive Service Forms (ISF) available.

The reserved dictionaries list all available personnel information, and allows to designate which attributes should be part of each dictionary. Choose the attributes to be included in the dictionary by checking the corresponding attribute name. For example, include Supervisor information, since some service requests may require the customer's supervisor's approval. Attribute names and data types cannot be changed. Ensure that you include any attributes that need to be manipulated in a service, even if the field will always be hidden. Choosing the attribute ensures that it is populated with the corresponding value from the person's profile. Designers can then configure the form containing the dictionary to hide the field as appropriate.

The personnel profile includes 10 custom fields (named Custom1 through Custom10) that are not used by Service Catalog. Any of these fields can be included in the Customer or Initiator dictionary. Some of these fields may be mapped to person attributes imported via directory (LDAP) integration. Others may be reserved for assignment or manipulation via the Display Properties or Conditional Rules available when configuring the Customer-Initiator Active Form Component which includes this dictionary.

For a Person Based dictionary and reserved dictionaries (Customer and Initiator dictionaries), you use the Dictionaries component of Service Designer to choose which dictionary fields (attributes) will appear on a service form. Simply navigate to the dictionary, and check or un-check the check boxes in the "Use" column to determine which fields are used, or included, on a form that uses the Person Based dictionary, or on the Customer-Initiator form—the active form component which automatically includes both Customer and Initiator dictionaries.

In addition to a customer or initiator's first and last name, and login ID, it is common to add fields for the person's email address, and home OU. The Person_ID is the unique identifier assigned to the person. Basic form processing does not require use of this field; however, it may be useful in writing data retrieval rules, for example, to dynamically retrieve additional information about the person or his/her supervisor that does not have a corresponding attribute in the dictionary. Similarly, the Supervisor_ID is also selectable for inclusion in the reserved dictionaries. This would allow the service designer to dynamically construct a "chain of command" for approval of chosen requests.

Similarly, you may want to include location information, especially in the Customer dictionary. This information may be required, for example, to determine which queue a task should be routed to, or simply to indicate the person's address, in case contact via an actual physical visit is required.

Automatically Included Integration Dictionaries

The Integration dictionary group is automatically created in all Service Catalog instances. Any dictionary that is created through the Integration Wizard (Service Designer's wizard for creating web services integrations between Service Catalog and external systems) is automatically placed in this group. Once created, integration dictionaries can be moved to any dictionary group. Designers cannot manually place dictionaries in this group.

Creating a Dictionary Group

A dictionary group is a folder that helps organize potentially large numbers of dictionaries. All dictionaries must be placed into a dictionary group. No orphan dictionaries are allowed by the system. Dictionary groups have no behavior associated with them, unlike service groups.

Step 1 Choose **Service Designer** > **Dictionaries**.

Step 2 Choose **New** > **New Dictionary Group**. The New Dictionary Group window opens.

Step 3 Enter the required information in the fields provided and click **Add This Dictionary Group**. The dictionary group is added to the tree menu.

Creating a Dictionary

Before You Begin

- Decide the type of dictionary. Understand the types of dictionaries available in Prime Service Catalog. See Types of Dictionaries in Prime Service Catalog, on page 2.
- For creating external dictionaries, define a data source name for the external database depending upon your application server, so that it can be used as the system dictionary. This is done completely outside of the system.

Step 1 Choose **Service Designer** > **Dictionaries**.

Step 2 Choose a previously defined dictionary on the left, or choose **New** > **New Dictionary** to create a new dictionary.

Step 3 Specify the type of dictionary.

a) To configure Internal Free-Form Dictionaries, click the **Free Form** link in the Add New Internal Dictionary section below the Data Source column.

b) To configure internal person-based dictionary, select **Person Based** from the **Template Based** drop-down list in the Add New Internal Dictionary section below the Data Source column.

c) To configure service item-based dictionary, in the **Service Item** field of the Add New Internal Dictionary section, search for the service item by name, and then click on the service item from the popup window that appears.

d) To configure external dictionaries, in the Add an External Dictionary section, click the name of the data source based on which the dictionary should be created. The New External Dictionary window displays. The login and password for the data source have already been configured on your application server.

Step 4 Add data elements to the dictionary usingTable 2: Fields to Configure Internal Dictionary

Step 5 Add dictionary attributes using Table 3: Dictionary Attributes table. To add multiple attributes, click **Add Field**.

Step 6 Click **Save Dictionary**.

To delete a dictionary from the system, choose **Service Designer** > **Dictionaries**, select the specific dictionary you wish to delete and click **Delete Dictionary**.

Note

- While creating a dictionary, values in the Service Form are pre-populated if you have selected **Enable Typeahead** in the Display properties of Active Form Component.

  However, this is applicable only for fields that have DataType as Text and when Input Type field contains either single-select or multiple-select values.

- You are prevented from deleting a dictionary, or a dictionary field, if there are associations with any conditional or data retrieval rules.

Table 2: Fields to Configure Internal Dictionary

Field	Description
Dictionary Name	Enter a name for the dictionary in this field. Allowed characters include: alphanumeric, underscore, and apostrophe. No spaces are permitted.
Group Name	Click the **Update** button to place this dictionary in a Dictionary Group. There is no system behavior related to the association of a dictionary with a dictionary group.
Default Caption	Enter the text you wish to display on the service form section title in this field.
Contact Person	This is the individual who should be contacted to discuss any potential modifications to the dictionary or its use within the context of the organization's site standards and best practices.

Field	Description
Service Item Family	This is an optional text entry field which enables grouping of multiple dictionaries to create a logical entity for the purpose of reporting. For example, three separate dictionaries together may constitute a complete set for Desktop Configuration. **Note** This field is not applicable for external dictionary.
Reportable	When a dictionary is reportable, the ability to modify the dictionary definition is temporarily disabled. The data types of fields in reportable dictionaries cannot be switched between numeric/money, date/datetime, and the character types. If you would like to make any other change to the dictionary, you can switch the Reportable setting to “No”, save the dictionary, make the changes, and then restore the setting to its previous value. Set the Reportable field to “Yes” to add the dictionary to those that can be reported against using the Advanced Reporting Ad-Hoc reporting and Report Designer features. If you specify that a dictionary is reportable, then you will see it as a query subject in Ad-Hoc reporting or Report Designer. **Note** There are many implications to be considered in the decision to make a dictionary reportable. See the Cisco Prime Service Catalog Reporting Guide for guidance in the decision to make a dictionary reportable. If you mark an external dictionary as reportable, please be aware that not all data types are supported.
Description	Enter a description of how this dictionary is to be used. This is very helpful information for other service designers who might use this dictionary.
Revision Notes	Ignore this field at setup; later, when making modifications to this dictionary. You can enter revision notes and click **Save Dictionary**.
DBA Notes	Use this field, optionally, to enter database-specific information about the use of this dictionary. This text area is most relevant for external dictionaries, since these connect to external tables which may be maintained by a DBA other than the Request Center DBA.

Field	Description
Dictionary Represents	This field is applicable for configuring External Dictionaries only. Click a radio button to choose either the data in a table (read/write) or the results of a query (read-only). If you click the data in a table (read/write), do the following: **1** Choose the table name from the Table Name drop-down menu. **2** Choose the column in the data table that has been reserved for storage of the requisition number from the Requisition Entry ID Column drop-down menu. **3** Choose the primary search key from the Primary Key Column drop-down menu. If you click the results of a query (read-only), do the following: **1** Enter the SQL statement in the SQL statement field. **2** Click Test SQL Statement to verify the statement.

Table 3: Dictionary Attributes

Field	Description
Name	Name for the field can consist of alphanumeric characters and the underscore and must start with a letter. It cannot contain spaces or other special characters. Reserved words in JavaScript (such as “this”) cannot be field names.
Type	Data Type influences the HTML representation that can be chosen when the dictionary is included in a form. That, in turn, influences how rules and ISF functions may be applied to the field, as well as the usage of the field within the Data Mart, should the dictionary, or a service which includes that dictionary, be made reportable. Choose the data type for the field. See theTable 4: Field Types table for a description of data types and their behavior.

Field	Description
Maximum, Decimals	Enter the maximum length of a string the user can enter into this field. For a number field, enter a number for the number of decimal places to the right of the decimal point. Enter 0 for an integer. For Service Item-Based Dictionary (SIBD), if **Encrypt** is enabled for an attribute the maximum length of a string the user can enter into this field is divided by 2. This is because the SIBD attributes insert form data into a Service Item table and during encryption the extra characters are inserted into the attributes.
Multivalue	Check the check box for Multivalue when you intend to use the INCLUDES operator in an expression to control the execution of a task. The INCLUDES operator compares the values of multivalue data types with other expressions. Multivalue fields map to the following elements: Multi-Select, Checkbox, and Radio button. **Note** DefMultivalue and TxMultivalue records are created for multivalue fields.
Show in Grid	Fields set to “Show in Grid” and “Use” in a dictionary will display as columns in the grid on a service form when the dictionary is set to “Display as Grid”. “Show in Grid” and Multivalue are mutually exclusive, as a grid does not have the capability for a multivalue cell. Grids cannot have more than 20 columns by default, which is defined by “dictionary.attributes.maximum.showingrid.count” property in the newscale.properties file. **Note** Fields in External dictionaries cannot be used in grids and hence do not have the “Show in Grid” column. Also, fields in Reserved dictionaries (Customer_Information and Initiator_Information) cannot be used in grids because those dictionaries inherently represent only one set of data. See Designing Grid Dictionaries for Fields with Multiple Data Instance, on page 31 for more information on grids.

Field	Description
Encrypt	Select the checkbox to encrypt sensitive data such as passwords, beneficiary names, account numbers, and so on. Sensitive information could be exposed while it is being stored, viewed, accessed, logged, or sent to external system and is automatically masked when you encrypt during configuration. The encryption is applicable in the following locations: • Form Data in Service • Service Item Manager > Manage Service Items • Service Catalog > My Stuff • Administration > Utilities > Form Data Viewer • Service Link > View Transactions > Message Type link > nsXML Message / External Message • My Services > Service Items • Server Logs • nsAPI to Read SI data • HTML Source. **Note** You can configure a maximum number of three secure strings for a dictionary. This restriction is because during encryption the resultant value contains some extra characters than the original field value which may impact performance. Therefore, while configuring the maximum length of a dictionary, provide some buffer length to ensure that the encrypted value doesnot exceed the length of dictionary specified. Secure Strings are applicable only for Text field types.

Table 4: Field Types

Type	Description and Active Form Implications
Text	Default data type, supports alphanumeric data; should be used for fields to be rendered as single- and multi-line text.
Number	Data type for all numbers, including integers and whole numbers; it is critical to specify the precision in the Decimals column, as the application will validate decimal precision as well as length.

Type	Description and Active Form Implications
Account	Documentation only; data treated as alphanumeric.
Date	Data compatible with the database's native datetime type, but display restricted to the date; calendar widget supplied for data entry.
Boolean	Data object whose possible values are “true” and “false”, presented as “Yes” and “No”.
Phone	Documentation only; data treated as alphanumeric.
SSN	Documentation only; data treated as alphanumeric.
Money	Data validated to contain only a valid number; monetary symbols or commas cannot be typed; accepts numerical characters and up to 3 decimal places.
Person	Data validated against a person ID in the personnel profiles; a Person Search dialog box is available via a “Search” button automatically rendered on the service form. The Person data type is provided primarily for backward compatibility; if this capability is required, a person-based dictionary should be created.
URL	Data stored as alphanumeric; a saved value is represented both as text and as an HTML representation of the value, providing a link to the specified URL.
Date and Time	Data stored as a date and time; calendar widget supplied for data entry contains a time-selection widget.

Defining Permissions to Edit Dictionaries During Service Requisition

When a dictionary is used in a form, you may specify how access to the dictionary is controlled. Using this procedure you can define permissions per system moment to control which users may view or edit dictionaries in the requisition life cycle. Only those dictionaries in which the initiator must provide the details of the request are typically editable. Any dictionary used solely by approvers, reviewers or task performers have no access assigned. A request may have zero or more Authorization moments, depending on which authorizations and reviews have been configured for that service.

Implications of View capability for a Dictionary

Specifying that a dictionary can be viewed, but not edited, during a particular moment has the effect of rendering all fields in the dictionary as HTML labels (read-only text).

- The text for fields that can have multiple selections (check boxes, multi-select drop-down lists) will show any options previously chosen for those fields as a comma-separated list of values.
- The label for a Person field shows the name of the person previously chosen. The accompanying Search button is disabled. A second (hidden) object contains the unique identifier of the person chosen.
- URLs are rendered as an HTML label that is hyper-linked.

Since HTML DOM objects (that is, <input> tags) do not exist for fields in non editable dictionaries a limited subset of rules or ISF functions can be applied. These limitations are documented in the detailed discussion of conditional rules and ISF functions that follows.

The access levels are applied at the dictionary-level. In a particular moment, the entire dictionary, comprising all its fields, is either hidden, displayed, but with all its displayable fields rendered as boilerplate text; or displayed with its fields rendered as a set of HTML input objects which can be edited by users or manipulated by rules.

All access control assignments pertain to all tasks performed as part of the delivery plan. If you need to assign different permissions to dictionaries for different users in different tasks, you will need to do this via a conditional rule. The **Access Control** setting should always grant all permissions that are required on a particular dictionary for a particular user/group. Conditional rules can remove access privileges, but cannot grant them if they are not originally specified via the **Access Control** tab.
You can grant the following types of permissions:

Using Access Control Tab for Assigning Permissions

Assigning permissions via the Access Control tab has important consequences for designers specifying active form rules and for ISF programmers:

- **For hidden dictionaries:** Hiding a dictionary via **Access Control** means that no objects defined in that dictionary are present in the service form for the moments and participants for which the dictionary is hidden. Therefore, it is not possible to use rules or ISF to manipulate dictionary or field contents or visibility.

If you need to manipulate the values of fields in a dictionary that are hidden from the user, you can configure the form rules that act upon these dictionaries to be executed on the server side. For more information, see the Defining Form Behavior for Service Item-based Dictionaries, on page 56. For example, you can populate a set of fields with default values without the user being aware of those values.

- **For view-only dictionaries**: Fields in such dictionaries cannot be made editable. However, rules or ISF can be used: to get the current value of a field in a dictionary set to view-only by Access Control; to temporarily hide the field from view; or to apply other functionality provided by the framework.
- **For editable dictionaries**: Full capabilities are available to manipulate the appearance and behavior of the dictionary or fields within the dictionary.

Setting a dictionary's Access Control to “None” (that is, with neither View nor Edit checked) prevents the dictionary from being sent to the browser. However, you can still configure form rules to store data in such a dictionary, provided those rules execute on the server (see the Defining Form Behavior for Service Item-based Dictionaries, on page 56). This is a powerful technique for keeping the service form, as rendered in the browser, as small as possible, while still collecting and storing the results of the browser session in the database.

Administrative Override of Access Control Settings

Access controls assigned via Service Designer are ignored when the service form is run by a user who has been granted the “Manage Service Dictionaries” capability. (This capability is included in the “Site

Administrator" and "Distributed Service Designer" roles.) For such users, all dictionaries are editable in all moments, regardless of the access controls specified.

Tip This capability is provided for ease of developing and testing the form appearance and should be assigned sparingly.

Before You Begin

See Implications of View capability for a Dictionary to understand the behavior of fields in the non-editable dictionary.

- See Using Access Control Tab for Assigning Permissions to understand the implications of assigning edit/view permissions to the dictionaries using Access Control option.
- See Administrative Override of Access Control Settings to see how these dictionaries permission can be overridden.

Step 1 Choose **Service Designer** > **Active Form Components**. Click a form from the Active Form Component tree.

Step 2 Select the form you wish to configure and click the **Access Control** tab.

Step 3 Click on a system moment and click on the dictionary you wish to configure.

Step 4 Click **View** and/or **Edit** for each participant in that moment.

Note If you do not check either View or Edit, then you are effectively granting no permissions to the participant for that dictionary during the specified system moment. This means the participant will not be able to see the dictionary fields at all during the specified system moment.

Step 5 Select the participant type from the **Add Participant** drop-down list. For more information about the participant types and its associated capabilities, see Table 5: Participant Types table.

Step 6 Continue by configuring permissions for all dictionaries during the chosen system moment. Move through the remaining system moments and dictionaries.

Step 7 Click **Save Form**.

You can easily remove a dictionary from an active form component if it is no longer needed, or you can delete a dictionary from the system, thereby removing it from any form components in which it was used.

Table 5: Participant Types

Participant Type	Description
Customer	The user who completes and submits the service order form; typically has View access to dictionaries in the Service Delivery moment, except those that contain confidential information

Participant Type	Description
Service Team	Service performers and managers who are members of the service team OU that "owns" the service group where this service resides. The "Service Team" participant refers to the service team that owns the service group in which the service resides. If you include this form in multiple services, the service team participant for each service may vary, depending on what group the service is in. Therefore, it may be preferable to "**Add Participants**" to the form, explicitly indicating the service teams that are needed to access dictionary data at this time. For services with complicated delivery plans, each task in the plan may need to be performed by a different service team. In this case too, additional participants should be added.
Organizational Unit	Members of the customer's Home OU, such as those who review or authorize service requests.
Financials Team	Refers to the OU, if any, which is configured in Administration as the Financials Team for the purpose of site-wide Financial Authorizations
Ad-Hoc Task Performers	Any performer, or queue members, who receive an Ad-Hoc task

Note The Customer-Initiator form presents additional concerns, since it will likely be used in virtually every service. Therefore, especially if the Customer and Initiator dictionaries should have View only permissions, it might be most efficient to "Add access for Anyone" via the **Additional Participants** option.

Adding Access Control Settings to a Specific Service

Access control that must be granted to a form component used in multiple services may need to vary on a service-by-service basis, depending on the service teams involved in the fulfillment of each request. Additional participants for viewing/editing dictionaries can be added to the service definition. To do so, choose Service Designer > Services > Form, select the dictionary, then add the appropriate Participants. No other changes can be made to Access Control settings at the service level.

Defining Standards in Service Design

Standards aid in the design of active form components, particularly in data retrieval rules that enable customers to drill down to specific answers or choices when ordering services. Standards provide reference data that can be used to validate user entries into a service form or to provide default values for fields on that form. You

can create new standards or import an existing standards data or definitions from a file using Service Item Manager in Prime Service Catalog.

Standards tables perform the same functions as relational database tables maintained in an external datasource—rows can be retrieved either for display (in a drop-down list) or validation (of user-supplied data). Standards tables and external tables differ in these ways:

- Standards tables can be maintained wholly within Service Item Manager—no DBA intervention is required to create the table, modify its structure, or maintain its contents. In addition to providing a user interface for creating, deleting or modifying Standards records, you can also import data from an XML file into a Standards table to create or modify the structure of that table.
- Standards tables can be used directly in table-based data retrieval rules, by choosing the standard as the datasource. They are also available for use in SQL-entry data retrieval rules or in the construction of SQL-based option lists.

Note By default, Catalog Deployer deploys any standards entries when a service deploy a data retrieval that uses the specified standard. You can override this behavior by changing the Administration setting to "Deploy standards entries". This would be desirable, for example, if administrators in the production environment were responsible for maintaining the standards, rather than having all standards defined in a production or test environment.

You can define permissions to view or edit service item standards based on the role assigned to a user. For more information on permissions that can be added, see Service Item Manager module in Assigning Permissions section in Cisco Prime Service Catalog Administration and Operations Guide.

To Define Standards

1. Specify the functional requirements for the Standard table. What data does each table need to contain? How will it be used within Service Catalog?
2. Use Service Item Manager to define the Standard. See Configuring Standards.
3. Populate the Standard table with data relevant to your Service Catalog setup. See Adding Data to the Standards Table.
4. Write data retrieval rules to access the Standard table.

Creating Standards Group

Procedure

Step 1 Choose **Service Item Manager** > **Design Service Items** > **Design Standards**.

Step 2 Click the plus sign (+) on the Standards Table on the left pane and select **New Standard Group**.

Step 3 Specify a Group Name and a description and click **Add**.

Adding Role Based Permissions for Editing Standards

Step 1 Choose **Service Item Manager** > **Design Service Items** > **Permissions**.

Step 2 The Permission Summary window lists the permissions assigned to the selected service item.

Step 3 Click the magnifying glass icon in the Role field, to select a role.

Step 4 Select a permission from the **Permissions To** drop down list and click **Add**.

Configuring Standards

Using this procedure you can create new standard definition and also edit the defined ones.

Step 1 Choose **Service Item Manager** > **Design Service Items** > **Design Standards**.

Step 2 Click the plus sign (+) on the Standards Table on the left pane and select **New Standard**.

Step 3 Add the attributes definitions and click **Add**. The attributes specify the fields (data) that are maintained about the standard. See Table 6: Attributes for Standards.

Step 4 To update existing standard definition, modify the attributes that were defined while creating Standards table and click **Save.**

Step 5 Click **Add in the Column Definitions** pane to add definitions to the table.

- **Display Name:** The name of the attribute that will appear as the attribute label on all Service Catalog pages, including the My Items portlet.
- **Name:** The internal name for the attribute; keywords and reserved words in the underlying database, for example, INTEGER, ORDER, or VARCHAR, cannot be used. Name length is limited to 27 characters.
- **Attribute Type:** The data type for storing attribute data.

All attributes added or updated since the last save are marked by a red triangle at the upper left of the attribute's display name.

If you have standards defined, you can also import these into Prime Service Catalog using **Import Data** tab in Service Item Manager. The details on how to import the standards from an external system, see Importing Standards.

Table 6: Attributes for Standards

Field	Description
Display Name	User-friendly version of the name; it may contain spaces.

Field	Description
Name	Standard name by which the system references the standard and its data. It corresponds to a table that is dynamically created and maintained in the Service Catalog transactional database. A Standard name can contain only alphanumeric characters and the underscore (_), with no embedded spaces. It must begin with an alphabetic character. Service Item Manager creates a database table with the same name as the standard name with a prefix of “St”
Standard Group	Select the group you want to associate the standard table with.
Description	A description about table you are creating

Adding Data to the Standards Table

Service Catalog supports the following ways to add data to standards tables:

- Use the **Manage Standards** tab in Service Item Manager to interactively edit standards data. This tab presents a grid containing all attributes specified for the standard.
- Use the **Import Data** tab in Service Item Manager to import standards data or definitions from a file or a service link. See Importing Standards for details on importing standards from a file or a service link. To know the format of the file that can be imported in Prime Service Catalog, see Import File Format For Service Items and Standards.

Importing Standards

Standards can also be imported using options similar to those available for service items—either the standard definition, the standard entries, or both can be imported. Service Item Manager includes the following methods for importing service items and standards:

- The Import Data option in Service Item Manager allows administrators to import service items or standards on demand from file. See ImportImporting Standards from File.
- A request can include a Service Link task to import service items or standards. See Importing Service Items and Standards using Service Link.

Importing Standards from File

Using Service Item Manager > **Import from File** option you can import data and definition for standards from an external source. The file to be imported must use ANSI encoding—either ASCII or UTF-8. Unicode encoding is not supported. You can select either **Data** or **Definition** option to import only those portions of the file, or you can select both. If you choose to import both and the XML file contains only a definition section, the system only imports the definition.

When importing standards data, all existing data is always overwritten by imported data.

Note The option to import standards data supplements or replaces Catalog Deployer's actions in deploying a service that includes a data retrieval rule that references a standard. By default, Catalog Deployer will deploy both the standard definition and any data previously defined in the source environment to the target environment. This behavior is desirable if standards data does not vary from environment to environment. If this is not the case, you may alter this default behavior by turning off the Administration setting to "Deploy Entries (data) in Standards Tables".

When importing a standards definition, the same conflict resolution options are available that are summarized below.

Table 7: Conflict Resolution for Standards Definition

Definition	Conflict Resolution	Description
Definition	Overwrite	If a standard record exists whose attribute values match all attribute values of the standard being imported then this standard is updated so that standard has values ONLY for the attributes that are specified in the record being imported. This implies that if the standard record being imported has specified values for only some attributes BUT the existing standard in the database has values specified for additional attributes, then those values would be set to NULL.
	Merge	Same as Insert below.
	Insert	If a standard exists whose attribute values match all attribute values of the standard being imported then this standard is not created.

Configuring Service Form Appearance and Behavior Using Active Form Components

A form is a building block for implementing a service. Each orderable service consists of one or more forms. Each form specifies the appearance and behavior of the web page presented to users when they order a service from the service catalog; authorize or review requests for services; and complete the steps required for delivery of the service to its recipient. That web page is called a "service form."

Form rules allow service designers to make service forms a Rich Internet Application (RIA) without having to write code. (RIA means that the application responds immediately to what the user types or what appears

on the screen, without the user having to click a “Submit” button at the end of a screen of input.) The rules allow designers to specify how the service form's appearance and behavior should change in response to user-initiated events.

Some common uses of conditional rules include:

- Enable/disable or show/hide fields based on a radio button (for example, Yes/No selections)
- Mark fields as mandatory based on the value of another field, or during a particular task or moment in the delivery (fulfillment) cycle
- Show/hide entire dictionaries to customize the user experience for different tasks within the service delivery moment
- Set focus on a field to attract the user’s attention
- Validate data for correctness

Dynamic Data retrieval rules provide an online, real-time interface between the service form and information stored in a relational database. These rules allow such data to be displayed in dictionary fields, or interrogated to determine the correctness of items entered by service requester or fulfiller. Some common uses of data retrieval rules include:

- Prefill form data with information maintained via other applications such as a Configuration Management Database (CMDB), ERP, or HR system
- Provide dynamic drill-downs so that the items listed in a drop-down list vary dynamically based on an item previously entered or chosen from another list

Creating an Active Form Group

All active form components must be placed into a form group.

Step 1 Choose **Service Designer** > **Active Form Component**.

Step 2 Choose **New** > **Form Group**.

Step 3 Enter a name for the new form group, and, if desired, a brief description and click **Save**.
The new form group appears at the top of the tree until the page is refreshed, where it is moved to its alphabetical location.

Assigning Group Level Permissions to Design Service Form

Assign object-level permissions for the form group using the Permissions tab. These object-level permissions are types of actions that people, organizational units, groups, functional positions, and roles are allowed to perform. Users can view only those forms groups on which they have permissions.

You can grant the following types of permissions:

- View Forms- Allows the users read-only view of all the forms under the group.
- Design Forms- Allows the user to design all forms under the form group and change group data. These rights are typically reserved for the individuals who design and configure forms in this form group.

- Create Forms- Allows the user to create forms and edit them.
- Maintain Forms-Allows the user to maintain all forms under the form group.
- Access Permitted Forms- Allows the user to view the form group and forms permitted under the form group.

You can grant the following form level permissions:

- Read: Grants read-only access to the forms.
- Read/Write: Grants edit access to the forms.

Note New form groups should not be set up to allow "Anyone" to design forms. The ability to "Grant access to Anyone" manually is a time-saving option intended for use once a form group is properly defined and established.

If you had an earlier version of Service Catalog, for each service group, a corresponding form group will be created. The name of the form group will be the prefix "UPGD: Form," followed by the service group name.

Creating an Active Form

Active form components are the building blocks of a service form, and dictionaries, along with active form rules, are the building blocks of a form component.The appearance and behavior of a service form is determined by how the dictionaries and their component fields are configured as part of the active form components that are used in the service definition.

Step 1 Choose **Service Designer** > **Active Form Component**.
Step 2 Choose**New** > **Active Form Component**.
Step 3 Enter a name and brief description for the new form.
Step 4 Click on the form group field and select one of the groups to associate with the form.
Step 5 Click **Save Form**.

After saving the form, you can add dictionaries to the form, and modify and configure form attributes. For more information on how you can add dictionaries and modify form attributes, see Configuring an Active Form.

Configuring an Active Form

Note Reusable form components should only be configured once for use across multiple services. When you modify the form component, the changes are applied to all services using the active form component.

The following attributes can be configured for an Active Form:

- **Form Content**: Dictionaries included in the form and the order in which the dictionaries and fields that comprise the dictionaries are displayed. For more information, see Adding Dictionaries to a Form, on page 25.
- **Display Properties**: How the individual attributes, comprising of each dictionary, are rendered on the web page when a user is working with a service form. For more information, see Defining the Appearance of a Form, on page 27.
- **Access Control**: Which users, or group of users, are able to view or edit specific dictionaries that comprise the service form at a each moment in the requisition life cycle. For more information, see Defining Permissions to Edit Dictionaries During Service Requisition, on page 15.
- **Active Form Rules**: Rules which can conditionally alter the appearance or behavior of the dictionaries or individual attributes displayed on the service form, or can dynamically retrieve data from relational data sources. For more information, see Configuring Dynamic Form Behaviors Using Form Rules, on page 33.
- **Active Form Behavior**: The events which trigger execution of a conditional rule or a script written using JavaScript in conjunction with ISF. For more information, see Adding Form Rules to a Service Form, on page 54.

Adding Dictionaries to a Form

The first step in configuring a form is to specify the dictionaries that are used in that form, the order and orientation of those dictionaries, and the fields in which each dictionary appears. Use the **Form Content** tab to add dictionaries. Only those dictionaries are displayed to the logged in user on which the user has dictionary or dictionary group level permission, or have inherited 'write' permission on dictionary from service or form.

Note When you view dictionaries included on a form on this tab, dictionary names are prefixed with the dictionary group name. Click the plus sign (+) to the left of the dictionary name to display the fields in that dictionary. To change the order in which a dictionary or field is displayed, click the component to move and then click the Up- or Down-Arrow keys at the right of the page until the component is in the desired sequence.

Before You Begin

Following are some design considerations before adding dictionaries to a service form:

- Choose how many, and which, dictionaries to include per form. Associate multiple dictionaries to a single form, only if it is for the same services. For example, if a group of services have a chain of approvals consisting of three approvers, to be chosen by the customer and dynamically determined based on data supplied on the request, the “Approvers” form should include three dictionaries: FirstApprover, SecondApprover, and ThirdApprover.
- Dictionaries do not need to be included in a form in order for rules defined in that form to refer to a dictionary or attribute. Further, a form may contain no dictionaries at all. In this case, the form is a repository for rules. The form must be included in a service that includes other forms which, in turn, include the dictionaries to which the rules refer.
- The same dictionary can be used in multiple forms, provided that only one of those forms is included in a service. However, all active components regarding form appearance or behavior would need to be defined in each form, so this is not an ideal architecture.
- For service bundle:

- Any rules or JavaScripts attached to form-level events (When the form is loaded or submitted) of the child services are ignored. A form component specifying these rules must be included in the parent service.
- If a dictionary occurs in multiple child services, it will appear only once in the service bundle. The dictionary's configuration matches the configuration specified in the first child service that includes an Active Form Component which, in turn, includes that dictionary. If the configuration of the dictionary is different in other services (for example, by the application of service-specific rules), those differences are ignored.

Step 1 Choose **Service Designer** > **Active Form Component**.

Step 2 Click on the form in the Active Form Components tree. Choose **Form Content** and click **Add Dictionaries**.

Step 3 In the Add Dictionaries dialog box, do the following.

a) In the **Name** column, search for available dictionaries and select the one you want to use.

b) In the **Display as Grid** column, check the dictionaries you want to display as a grid. Fields of the dictionary set to **Show in Grid** will display as columns in the grid. Only fields set to **Show in Grid** and **Use** will appear as columns when a dictionary is added with **Display as Grid** checked. Fields set to **Use**, but not to **Show in Grid**, will not appear.

c) Click **Add**.
The dictionaries are added to the form and returns to the Form Content page.

The **Display as Grid** check box on the Content tab appears checked for any dictionary added as a grid. The check box is always dimmed and cannot be changed on the Content tab; its purpose is to show which dictionaries on the form are in grid format. If you inadvertently add a dictionary as a grid, or forget to check **Display as Grid** when adding a grid dictionary to the form, you can remove the dictionary and add it again, with or without **Display as Grid** checked in the Dictionary dialog box.

Step 4 Check the options to show this dictionary when the active form component is used in a bundled service. **Show in Bundle** indicates the dictionary fields appear if the service in this form is attached to is part of a bundle.

Step 5 Click **Save Form**.
Services Used in This Form is a read-only table that automatically populates when the active form component is used by a service. For more information, see Defining Form Behavior for Service Item-based Dictionaries, on page 56.

After a dictionary is added to an active form component, you can configure form fields and form behavior.

If a form uses multiple dictionaries, you can change the order in which they appear on the form by checking the check box to the left of the dictionary name and using the Up- or Down-Arrow keys on the right of the page.

Note The dictionary names on the Form Content tab are a link to the Dictionaries component of Service Designer. If you need to add another field to a dictionary, you can simply Ctrl-Click on a dictionary name from here and go directly to the dictionary.

Defining the Appearance of a Form

The appearance of each active form component is configured on the **Display Properties** tab. On this tab you configure the properties of each dictionary and the fields within. A dictionary field's data type is assigned when a dictionary is defined, which also defines the field's storage requirements. For more information on fields that can be defined in a dictionary, see Creating a Dictionary, on page 9.

Step 1 Choose **Service Designer** > **Active Form Component**.

Step 2 Select the form you wish to configure and assuming there are dictionaries already added to the form, click the **Display Properties** tab.

Step 3 From the **Dictionaries Used in This Form** section, expand the dictionary whose fields you want to edit.

Step 4 Click on a dictionary name to display its properties on the right.

Step 5 Enter a caption for the dictionary, and click Change Caption, or Set to default caption. If the dictionary has been set to **Display as Grid**, enter the grid settings you want in the Grid Options section.

Step 6 Click on a dictionary field and configure the requirements. For more information, see Table 8: Fields in HTML Representation.

Step 7 Click **Save**.

Step 8 Repeat Step 1 to 7 for all dictionaries configured.

Note To change the order in which dictionaries and fields appear on a form component use the **Form Content** tab.

Table 8: Fields in HTML Representation

Field	Description
Input Type	For a grid dictionary, “Select (multiple)”, “radio”, and “checkbox” are not available in the drop-down list of Input Types, as these controls cannot be rendered in a grid. **Note** Depending on the Input Type chosen, the page will dynamically change the available configuration options. For example, you can choose a maximum or minimum number of characters allowed within a text field, but not for a password field. If the Input Type involves a choice for the end user (for example, radio, checkbox, select), you can also choose whether the choices appear vertically or horizontally on the form, and can configure the choices using the Options List and Options Preselection sub tab. For more information on each of the input types, seeTable 9: Input Types for HTML field Representations table.
Label	Edit the Label. For example, Change "**Select_Person**" to read: "Select Person"
Help Text	Provides direction to the end user as to how to complete the form. For example, "Please enter your 9-digit SSN without any spaces or dashes."
Default Value	Enter a value here if you want to use namespaces to prefill this field.
Generate Unique Value	You can generate a unique ID as the value for any dictionary field with an Input Type of “Text”, “hidden”, or “read-only” by checking the check box “Generate unique value”. When a new service is requested, a universally unique identifier (UUID) is generated and set as the value of the field upon form load for all fields checked “Generate unique value”. Conditional rules and data retrieval rules are executed after this step; so if there is a conditional rule or data retrieval rule that sets the value of this field, then the UUID value may be overwritten by the rule execution.
Validate Range	Check or uncheck this check box if you want the system to validate the user's input. Not all field types support validation.

Field	Description
Mandatory	Check the Mandatory check box to require the end user to complete the field before submitting their order.
Add a Button	Select Add a Button if you want to add a button to the form. Enter a fully qualified URL (including the http://) to take the user to another site when they click the button This control does not appear in a grid dictionary, because you cannot configure buttons in a grid
Editable on server-side	Check the Editable on server-side only check box to set the field to be edited only on the server. Dictionary fields that contain sensitive information, or contain values set by default or with auto-retrieval mechanisms should have this check box checked. This is a security feature that prevents hackers from intercepting and changing data in the browser.
Advanced Formatting	Additional or alternative HTML formatting can be applied to a specific field by using the Advanced Formatting button.This control does not appear in a grid dictionary, because you cannot configure Advanced Formatting in a grid.

Table 9: Input Types for HTML field Representations

Input Type	Description and Active Form Implications
text	Rendered as an HMTL “text box”.
textarea	Rendered as a multi-line text area.
password	Rendered as a password field, where values are displayed or echoed as asterisks.
hidden	Rendered as an HTML text box, but with a display type of “hidden”.
radio	Rendered as an HTML option list; the field has multiple options—the field value is the option currently chosen. Not available for fields set to “Show in Grid”.
select (single)	Rendered as an HTML drop-down box from which the user can choose a single value.

Input Type	Description and Active Form Implications
check box	Rendered as a set of HTML check boxes for all the possible options as designed in Service Designer. Not available for fields set to “Show in Grid”.
select (multiple)	Rendered as an HTML drop-down box from which the user can choose multiple values. Not available for fields set to “Show in Grid”.
SSN	Rendered as a single-line text box; “SSN” provides documentation only.
Person	The field is rendered as two objects: a drop-down list displays people and allows users to choose one; a second (hidden) object contains the unique identifier of the person chosen. This HTML representation is automatically applied to the Select_Person field within a person-based dictionary. It could also be applied to a field in a free-format dictionary with a “Person” data type; however, this latter usage is provided primarily for backward compatibility and is not recommended.
URL	Rendered as a single-line text box with a link generated for accessing the URL once the value has been saved.
read-only	Rendered as text—a user will not be able to enter data. The service designer is responsible for providing a value (typically via a form rule or default value display property) to such fields.

Input Type	Description and Active Form Implications
slider	Rendered as a slider component. The Slider component displays the values for text, boolean, money, or number. Also, it displays the maximum and minimum values for money, and number. • **Text**: Use this option to display the values in form of text. • **Number**: Us this option to display numeric values on the slider. Ensure that the **Enable Range Slider** checkbox is unchecked. If it is checked, the values are displayed as range of whole numbers. • **Number Range**: Use this option to display numeric values as range of whole numbers. Check the **Enable Range Slider** checkbox and enter the values for the **Minimum** and the **Maximum** fields. Also, If you want the slider to move only to the specified interval range, enter the value in **Interval** field. For example: if you enter the value 5, you can move the slider only at an interval of 5, 10, 15, 20, 25, and so on. • **Money**: This option is similar to the **Number** option, but you can specify both whole numbers as well as decimal numbers. • **Money Range**: This option is similar to the **Number Range** option, but you can specify both whole numbers as well as decimal numbers. • **Boolean**: Use this option to have only two values on the slider, **On** or **Off**.

Designing Grid Dictionaries for Fields with Multiple Data Instance

A grid is useful when you are designing forms that require multiple data instances of the same fields to be entered in one form. Rather than creating multiple sections of the same fields, you can create one grid with the fields. A grid enables you to create and configure only one dictionary rather than multiple to hold the multiple sets of the same fields.

While configuring a grid, the dictionary is turned ninety degrees so that field labels appear atop the fields. The grid cells are not truly individual fields when rendered in the browser, although they are stored as individual fields in the database.

Difference between grid and non-grid dictionaries

- The Input Types of Radio Button, Check Box, and Select (Multiple) cannot be used.
- Field labels appear as column headers. Advanced formatting cannot be configured for the field labels in a grid.
- Buttons cannot be used.
- Some ISF Dictionary- and Field-Level Functions cannot be used (see Configuring Dynamic Form Behaviors Using Form Rules, on page 33 for details).
- Grid fields cannot be chosen as the "Triggering Field" when creating a data retrieval rule.
- Grid dictionaries and their fields cannot be chosen as the "Triggering Condition" for conditional rules—they can only be "Action" targets.
- A subset of conditional rule Actions are supported; and these apply to the column as a whole, not to individual cells.
- Requisition API (RAPI) cannot be used to submit services containing grid dictionaries.
- The use of grid dictionary fields for Business Engine namespace and Service Link agent parameters is not supported.
- Server-side conditional rules are not supported on a grid.

To Design Grid Dictionaries

Step 1 In **Service Designer** > **Active Form Component** > **Form Content** > **Content tab** choose the fields you wish to have appear in the grid by selecting the **Show in Grid** check box in the dictionary's definition.

- For a new person-based dictionary, **Show in Grid** is checked by default for the Select_Person, Login_ID, Personal_Identification, Email_Address, and Home_Organizational_Unit fields.

Note "Show in Grid" and Multivalue are mutually exclusive, as a grid does not have the capability for a multi-value cell.

- Fields in External dictionaries cannot be used in grids and hence do not have the **Show in Grid** column.
- Fields in Reserved dictionaries (Customer_Information and Initiator_Information) cannot be used in grids because those dictionaries inherently represent only one set of data.

Step 2 Choose **Display as Grid** check box **for the dictionaries** to appear as a grid (except Reserved dictionaries). Only fields set to **Show in Grid** and **Use** columns will appear as columns when this option is selected.

Note The **Display as Grid** check box appears checked for any dictionary you have added as a grid and cannot be changed on the Form Content tab. If you forget to check **Display as Grid** when adding a grid dictionary to the form, you can remove the dictionary and add it again, with or without **Display as Grid** checked in the Dictionary popup window.

Step 3 Set grid options and display properties.

Configuring Dynamic Form Behaviors Using Form Rules

Active Form Rules are defined on the Active Form Rules tab, on the Active Form Components page. The application supports two types of Active Form Rules:

- Dynamic Data Retrieval rules retrieve data stored in a relational database and either display the values returned into fields on the current form, or validate data on the form against the retrieved results. Understanding how
- Conditional rules specify a set of conditions that must be met in order for a set of actions to be carried out. Although referred to as "conditional," these rules are actually applied whenever the specified triggering event occurs

Most of the rule definition is done "declaratively"—that is, rule wizards help you define the rules, by walking you through a series of steps. In most cases, you don't have to write any code at all, just answer a series of questions. When a user orders a service containing the form, the rules are retrieved and code that operates in the context of a web page is automatically generated.

Creating Conditional Rules

Conditional rules allow designers to alter the behavior and appearance of a service form. These are defined in the Active Form Rules tab on the Active Form Components page. After creating the rules, these rules should be attached to a triggering event for these rules to execute.

A conditional rule has the following components:

- The rule name and description
- Conditions under which the actions specified by the rule should be executed (including being unconditional)
- Actions to be taken if all conditions are true

Conditional rules provide a powerful mechanism for applying logic both during the "conversation with the end-user" (that is, during the browser session), and before and after that "conversation" (that is, server-side).

Using this procedure create rules and attach the triggering event to execute the rules.

Before You Begin

- Understand Browser vs. Server-Side Events. See Server-Side Data Retrieval Rule, on page 95.
- Understand how dictionary fields are represented and stored, and how fields in a grid dictionary differ form their non-grid counterparts. See Difference between grid and non-grid dictionaries.

Step 1 Choose **Service Designer** > **Active Form Component**.

Step 2 Select the active form component to which the rule has to be applied, and click **Active Form Rules**.

Step 3 Choose **New Rule** > **New Conditional Rule**.

Step 4 In the first page (Step 1) of the Conditional Rule wizard, formulate a condition by specifying one or more conditional clauses. Each component condition evaluates to true or false. Multiple conditional clauses can be combined, using standard relational rules. You can also make the conditional rule unconditional.

a) Enter a **Rule Name**. This name must be unique on the form component.
b) Click **Add Condition** and choose a type of condition. Depending on your choice, a slightly different .Condition Builder dialog box displays. When choosing dictionaries or dictionary fields, the dictionaries used in your form component are listed first, followed by all dictionaries available in the system. For more information see Table 14: Conditional rules Actions, on page 39 table and Best Practices/Guidelines for Constructing Conditions, on page 37.
c) After each condition is defined, click **OK** to save changes.
If necessary, click **Add Condition** to continue building the statement. When multiple conditions are combined in one rule, you must join them using Boolean operators (or clauses). You may also click on the parentheses () located at the beginning and end of each condition (making the box pink) to clearly define the statement in the full statement condition box below the conditions.

Note If the “is equal to” operator is paired with a person type, the value must be a number corresponding to the person's ID and not the person's name. Do not use “is equal to <blank>” to check for a zero or null value for Number and Money field types. Use “is equal to 0” instead.

d) After all conditions are added, click **Next**.

Step 5 In the second page (Step 2) of the Conditional Rule wizard, add actions that should be executed when the specified conditions are met.

a) Click**Add Action** and choose the type of action. For more information about actions, see Table 14: Conditional rules Actions, on page 39 table.
If the condition set up in the first page of the Conditional Rule wizard is not met, no action will occur. To set up an alternate action, you must create another conditional rule. If you are setting the value of a any field to show a date, the format must be: mm/dd/yyyy. For example, 10/31/2010.

b) After action is defined, click **OK** to save changes and click **Next** to go to the next page.

Step 6 In the last page (Step 3 of 3) of the Conditional Rule wizard, review and save the conditional rule you created. If you need to make edits, click **Previous** to move back through the wizard, or click **Save** and edit the rule later.

a) In the Automatic Associations portion of this page attach the rule to one or more triggering events:

- **Before the form is loaded (server-side):** This server-side rule is executed on the server prior to sending it to the browser.
- **When the form is loaded (browser-side):** This rule is executed when the service form for any service containing this Active Form Component is initially displayed (loaded) in the browser.
- **When the value of any field referred to in the rule's conditions is changed:** In this case, the rule will be attached to either the “When the field is changed” or “When the field is clicked” event, depending on the HTML representation of the field.
- **After the form is submitted (server-side):** This server-side rule is executed on the server after the form is submitted on the browser and sent to the server for processing.

Note Always use the “After the form is submitted (server-side)" triggering event, when the form rule is used, to set the values of form fields that should be strictly controlled based on entitlement or access permissions. This server-side execution of set value action overrides unauthorized data that may be sent to the server through malicious attempts

To edit an existing rule, review or modify the events to which the rule has been attached on the Active Form Behavior tab.

Examples of a Conditional Rule

Here are some conditions that might be useful in a service form, with the actions they might entail:

Table 10: Examples of conditional rule

Condition	Action
Is the current value of the DatabaseType field in the Database dictionary equal to “Other”?	If so, display the Description field in the same dictionary, and require the user to enter additional information.
Are you ordering this service on behalf of someone else?	If so, display additional information about that person (the actual customer for the service).
Is the current user a task performer working on the “Order Memory” task?	If so, display the MemoryDetails dictionary and make entry of all fields required.

Note Grid dictionaries and their fields cannot be chosen as the “Triggering Condition” for conditional rules.

Table 11: Conditions in the Add Conditions list

Parameter	Description/Usage
Dictionary Field	Compare the current value of a field in a dictionary on the service form to a literal field, to the value of another dictionary field, or to a blank or null value. For details on the available operators, see the section on Operators below.
Dictionary	Determine the usage of the specified dictionary in the service form at the current time. A dictionary may be viewable, editable, or hidden (previously hidden by a conditional rule or ISF).
User Name	Compare the login name assigned to the current user to the specified value.
User Role	Check whether the current user has been assigned or not assigned the specified role.
Moment	Check whether the request is in a specific moment in the requisition life cycle. A requisition is also known as service request order. Table 12: Requisition Lifecycle States and Its Values list the values representing requisition life cycle states.

Parameter	Description/Usage
Task Name	Compare the name of the current task in Service Manager to the specified Task Name. The Task Name is set for all authorization and service delivery moments. The task name is blank for moments when a task is not relevant (“Ordering” and “Service Complete”) and when the service form is viewed in Service Catalog, regardless of the current status of the requisition.
Context	The module in which the service form is currently displayed. Valid values are: • Service Catalog • Service Manager • My Services
Service Name	The name of the service
Does unsubmitted requisition exist	A Boolean that returns **true** once the service request has been saved. This is **false** only when the service is initially requested, in the ordering moment. Once the request has been “Added” or Submitted”, it exists.
Is Order on Behalf	A Boolean that returns **true** if the service has been ordered on behalf of another user, and **false** if the requestor is ordering the service for their self.
Customer Role	Check whether the customer for the service has been assigned or not assigned the specified role.

Table 12: Requisition Lifecycle States and Its Values

Service Request Life Cycle Status	Parameter Value
Ordering	ordering
Departmental Authorizations	ouauthorizations
Departmental Reviews	oureviews
Service Team Authorizations	stauthorizations
Service Team Reviews	streviews
Financial Authorizations	financialauthorizations

Service Request Life Cycle Status	Parameter Value
Service Delivery	servicedelivery

Best Practices/Guidelines for Constructing Conditions

The Context Condition: In some cases, the Context may seem redundant—for example, a request is only visible in the Ordering moment in My Services. However, a service form may be viewed in the Service Delivery moment within Service Manager by task performers and within My Services by the user who originally submitted the request. Similarly, a form may be viewable by multiple participants in the Service Delivery moment. You may want to vary the form's behavior or appearance depending on who is doing the viewing, and where.

The Does Unsubmitted Requisition Exist Condition: When a requisition is initiated, it is in the Ordering Moment. At this time any rules or ISF functions that "prefill" form fields with default values are typically executed. The end user may complete data entry and immediately click Submit. In that case, assuming all data in the form is valid, the requisition goes from the Ordering Moment to next moment defined for that service.

In certain circumstances the user may save the requisition without submitting it by clicking "Add and Review". For example, a user may save a requisition if not all data for the requisition is available; if additional services must be added to the same requisition; or if an attachment must be added. The user may edit an unsubmitted requisition by choosing it in My Services.

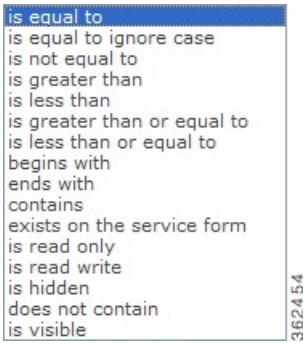
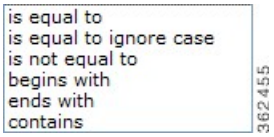
When an existing requisition entry is opened for edit before the requisition is submitted, the "**Does Unsubmitted Requisition Exist**" condition is true; it is false at all other times. Use this condition, for example, to ensure that a rule that prefills default values, is executed only for new requests, not for any requests that have not been saved. This ensures the data the user has previously supplied is not overwritten.

The Moment = Service Delivery Condition: The Service Delivery Moment may encompass many tasks. If a rule is to be conditionally executed for a particular task, use the Task Name condition.

The Task Name Condition: Be careful when referring to a Task Name, since this is a descriptive field that can freely be changed in Service Designer. This problem is minimized if service design guidelines are developed and strictly enforced for naming tasks.

Service designers sometimes include namespaces in task names. For example, the namespace #Name#, referring to the service name, is used to differentiate the same task when it occurs within multiple services. The Task Name used by a conditional rule has all Namespace references properly evaluated. String operations (only checking the beginning or end of the task name, or checking for a phrase contained within the task name) can be used to compare against the portion of the task name that is not derived from the namespace.

Operators in Conditions: The available operators are context sensitive, displayed in a drop-down list that depends on the condition you have chosen. For example, since a field may contain numeric, alphanumeric, or date data, both arithmetic and string operations are allowed, as well as operators to determine the usage of the field. Task and service names are text, so only string operations are appropriate. For conditions that are either true or false, or for which only a limited number of options are possible (such as the current context), radio buttons are available.

Dictionary Field	Task and Service Name	Is Order on Behalf/Does Unsubmitted Req. Exist
		

Most of the operators that can be applied to dictionary fields check the value of that field, comparing it to the value of the specified or literal field. Of these operators, only “is equal to ignore case” is case-insensitive. All other operators are case-sensitive; be sure to take this into account when writing rules such as “starts with” and “contains”, which operate on alphanumeric data.

Other operators, such as “exists on the service form” and “is read only” refer not to the value of the field but to its usage. You do not need to use these rules under most circumstances, since the runtime rule framework automatically checks for the presence of a field on the form and its usage before attempting to execute a rule.

Variables in Conditions: The value of any field in any dictionary may be used in a condition. Since the dictionary may not be included in the current form, the service designer must ensure that forms containing all dictionaries referred to in the rule are included in the service.

Combining Conditions within One Rule: Multiple conditions can be combined and evaluated to determine if the rule’s actions should be executed. In this case, the rule must include Boolean operators (AND, OR) and may include parentheses to alter the normal precedence of these operators.

The conditions essentially build an “if” clause. The Rule Builder does not currently support an “else” clause. If you need to apply if/then/else logic, define two rules with conditions that are mutually exclusive:

Table 13: Conditions versus Actions

Condition	Actions
Rule 1: DatabaseType_OtherDatabase.DatabaseType is equal to Other	Display the Description field in the same dictionary. Make the Description field mandatory.
Rule 2: DatabaseType_DefinedDatabase.DatabaseType is not equal to Other	Hide the Description field. Make the Description field optional. Set the value of the Description field to blank.

Actions are summarized in the table below:

Table 14: Conditional rules Actions

Action	Description of the Action	Notes	Grid Use
Show/Hide	Show/Hide the specified field, grid column, or all fields in the specified dictionary. The elements are displayed with the representations defined in Service Designer.	A frequent use of conditional rules is to show or hide fields or dictionaries based on the current value of another field on the form or any other conditions specified. If you need to hide most, but not all fields in a dictionary, you can hide the dictionary then show only the desired fields. If a field is hidden by a rule, its value is still accessible in other conditional rules.	Applies to entire column

Action	Description of the Action	Notes	Grid Use
Set Value	Set the value of a target field equal to the value of the specified source field, literal, null/blank or the result of an expression. When you set a field value, you need to take into account the field's data type and HTML representation.		Not currently available; use ISF instead

Action	Description of the Action	Notes	Grid Use
		If the value is set to a literal or the result of an expression, the literal or expression can include any numeric or text fields used in the current service. (Date arithmetic is not supported.) The field is represented in the expression using lightweight namespace syntax, that is, *#DictionaryName.FieldName*#. If the expression is invalid (for example, it includes division by zero), the expression is skipped; the value is not updated; and execution of remaining actions or rules for the same event, if any, continues. No error message is displayed to the user. The “Editable only on server-side” setting for a field also affects the behavior of Set Value. A field marked as “Editable only on server-side” is secured against any attempt to intercept its value during the browser session. In the case of person-based dictionaries and service item-based dictionaries with automatic retrieval enabled, the attribute values retrieved from the database will always override the values sent from the browser when the “Editable only on server-side” setting is enabled. For that reason, any Set Value action you wish to apply to such a	

Action	Description of the Action	Notes	Grid Use
		field must be in a rule tied to the post-Submit event—in other words, you can use Set Value to edit the field, but only on a server-side event.	
Set Price	Set the service price or the price of the specified child service within a bundle equal to the value of the specified source field, literal or the result of an expression.	The Set Price action supports dynamic pricing of a service. The price may be set to a constant, the value of another field, or the result of an expression formulated using the same rules as for the Set Value action. Although the Set Price action can be included in a rule triggered by any event, the new price actually takes effect only when the service form is submitted; if the transaction is cancelled, the new price is not recorded. Because the new price takes effect only when the form is submitted, the most appropriate event you should choose for executing a rule containing Set Price is post-Submit. The Set Price action is recorded in the requisition's System History.	Not supported

Action	Description of the Action	Notes	Grid Use
Make Mandatory	Make the specified field mandatory. Making a field mandatory has the same effect as designating a field as mandatory via Service Designer.	• The mandatory symbol appears to the left of the field label. • The user must enter a value in order to successfully submit the form. • If the administrative settings include the form monitor, the monitor will show the dictionary as incomplete until a value is supplied. If you are hiding a field (or dictionary), you must ensure that no fields in the dictionary are set to mandatory; if they are, an error message appears when users try to submit the form. If a rule (or ISF) is used to toggle the field between mandatory and optional, the field's HTML representation should define the field as optional. Using a conditional rule to override a field already marked mandatory in a HTML representation may result in unexpected behavior. If a mandatory field will be toggled between read/write mode, the field should be made optional at the time when it is set to read-only, or disabled to avoid possible confusions to the users.	Not supported
Make Optional	Make the specified field optional.		Not supported

Action	Description of the Action	Notes	Grid Use
Make Read-Only	Make the specified field, grid column, or all fields in the specified dictionary read-only; the field or column cannot be changed by the user, but its value can be changed by a rule or ISF.	In the case of a grid, there is no difference between “Make Read-Only” and “Disable”.	Applies to entire column
Make Writable	Make the specified field, grid column, or all fields in the specified dictionary writeable.	This action is identical to “Enable”	Applies to entire column
Enable	Make the specified field, grid column, or all fields in the specified dictionary writeable.	This action is identical to “Make Writable”.	Applies to entire column
Disable	Disable the specified field, grid column, or all fields in the specified dictionary. Disabling a field or column makes it read-only, like the Make Read-Only action; however, unlike Make Read-Only, it also dims the field or column, and ignores any changes applied to the field via a conditional rule or ISF.	In the case of a grid, there is no difference between “Disable” and “Make Read-Only”.	Applies to entire column

Action	Description of the Action	Notes	Grid Use
Set Focus	Move the cursor to the specified field. This is typically used after an alert or when the form submission is stopped, to direct the user's attention to a problematic field.	Set Focus does not work for flat dictionary right after grid dictionary. Follow these pattern to Set Focus for flat dictionary, • Do some input in the grid dictionary. • Click any field in the flat dictionary. • The first click will not set the focus. • Have to click a second time to set focus on that filed in the flat dictionary.	Not currently available; use ISF instead
Alert	Display an alert box with the specified message; only a literal message can be used—namespaces are not supported. The action does not apply to server-side events in general.	The only valid use of alerts on the server side is when it is coupled with the Stop Submission action in the post-Submit event.	N/A

Action	Description of the Action	Notes	Grid Use
Stop Submission	Does not allow the form to be submitted or updated. Should be used only in conditions where a triggering event is to submit the form.	When the action is executed on the browser side, it does not stop execution of any other actions in the rule, regardless of its sequence in the actions. When the action takes place on the server side, all subsequent actions and rules are skipped and an error message is presented to the end user. An alert action can be defined optionally prior to the stop submission action to provide the appropriate error message. If no alert message is defined, a generic error message is shown to state that the request cannot be submitted.	N/A

Conditional Rules and ISF for Grid Dictionaries

Grid dictionaries are different from nongrid dictionaries in the way they are rendered in the browser and stored in the WDDX generated in the database. Therefore you will need a mixture of conditional rules and ISF functions to manipulate individual cells. This section covers the conditional rule actions and ISF functions that can be used on grids.

Conditional Rule as Applied to Grids

Grid dictionaries and their fields cannot be chosen as triggering conditions for rules; they can only be chosen as the targets of conditional rule actions. The following conditional rule actions which apply to individual fields in non grid dictionaries, apply to an entire column in a grid. All other conditional rule actions are not supported on grid columns or cells.

- Show Fields
- Hide Fields
- Make Read-Only
- Make Writeable
- Enable
- Disable

In case of a grid, there is no observable difference between the Make Read-Only and Disable actions (whereas there is a slightly different user interface effect seen in non grid fields). The same is true for the Make Writable and Enable actions.

ISF in Grids

JavaScript functions cannot be assigned to any field-level event within a grid dictionary. The existing ISF framework has been extended to support grid dictionaries and fields, as described in the tables below.

Table 15: Dictionary-Level Functions:

Function	Usage in Grid
serviceForm.dictionaryName.setVisible (Boolean)	Hides or makes visible the Grid. Has no effect on a dictionary which is hidden via Service Designer.
serviceForm.dictionaryName.isVisible()	Returns true if the Grid is visible and false otherwise.
serviceForm.dictionaryName.getCaption(Boolean stripTags)	Gets the title text for the grid, optionally stripping any HTML.
serviceForm.dictionaryName.setCaption(String newCaption)	Sets the title text for the grid.
serviceForm.dictionaryName.setReadOnly (Boolean)	Sets all the columns in the dictionary to be read-only or read-write; has no effect if in Service Designer the columns were already set to read-only.
serviceForm.dictionaryName.isReadOnly()	Returns true if the dictionary is read-only.
serviceForm.dictionaryName.getGridSize()	Returns the number of rows in the grid.

Table 16: Field-Level (Grid Column-Level) Functions

Function	Usage in Grid
serviceForm.dictionaryName.fieldName. setReadOnly (Boolean)	Makes the column read-only or read-write.
serviceForm.dictionaryName.fieldName.isReadOnly()	Returns true if the column is read-only and false otherwise.
serviceForm.dictionaryName.fieldName.setVisible(Boolean)	Makes the column hidden or visible (displayed). If the column is hidden via Service Designer settings, it cannot be made visible.
serviceForm.dictionaryName.fieldName.isVisible()	Returns true if the column is visible and false otherwise.

Function	Usage in Grid
serviceForm.dictionaryName.fieldName. getInstructionalText(stripTags)	Returns the instructional text for a column, optionally stripping any HTML from the text based on the Boolean stripTags argument.
serviceForm.dictionaryName.fieldName.setInstructionalText(text)	Sets the instructional text for a column.
serviceForm.dictionaryName.fieldName.getPrompt(stripTags)	Returns the column header—optionally stripping any HTML from the prompt based on Boolean stripTags argument.
serviceForm.dictionaryName.fieldName.setPrompt(prompt)	Sets the column header.

Creating Dynamic Data Retrieval Rules

Dynamic Data retrieval rules retrieve data stored in a relational database and either display the values returned into fields on the current form or validate data on the form against those values. They perform this retrieval by executing a SQL query against the source database, returning the values of the columns you have specified to the service form.

Note When a service is ordered through web services, only server-side rules will be executed. If there is any selected field that has options populated by data retrieval rules, the rule should be triggered after submission to ensure that the value passed in the web service request has a list of options to validate against.

Prerequisites to Creating Data Retrieval Rules

- **Define Datasource:** In order for Service Catalog to access a database, a corresponding datasource must be defined. A datasource identifies the database and supplies information for connecting to it, including a valid user name and password. The application comes with one datasource preconfigured, named REQUESTCENTERDS, which provides access to Service Catalog data. A database administrator together with a system administrator needs to define any additional datasources, containing company-specific data, and publish the definitions to the application servers on which Service Catalog is installed. Detailed instructions for configuring and installing datasources are given in the Cisco Prime Service Catalog Installation and Upgrade Guide. Standards and Service Items can be used as a source of data in data retrieval rules. However, their usage is restricted to a rule with the Query Type of "Database Table" are not available for use in rules that use SQL.

When you define a database table lookup, the list of datasources includes Standards (and the Service Items), as well as the transactional and Data Mart databases. If you choose "Standards", the available Standards tables appear that can be chosen as the Table Name for the query. All other aspects of composing the data retrieval rule are identical to those explained previously in this section.

- **Identify the Structure of the Source Data**: You will also need to know the structure of the data you need to retrieve, and tables in which it is stored. An IT specialist knowledgeable in the source system can provide this information.
  - If all of the data you need to retrieve resides in a single table, you can simply specify the name of the table and columns to retrieve. Service Catalog will automatically build a SQL query which retrieves all columns in the table.
  - If you need to retrieve data from multiple tables or to manipulate the values (for example, concatenate values together or perform calculations) for use on the service form, you will need to write and test a SQL query yourself. A database or IT specialist with knowledge of the source system and access to tools for developing SQL are indispensable for this task. Once the query is tested, you can cut and paste it, with minor modifications outlined here, into the Dynamic Data Retrieval Rule Wizard using a Query Type of "Enter Your Own SQL Query".

Use this procedure to define dynamic data retrieval rules.

Before You Begin

- Define the datasource and identify the structure of data to be retrieved as described in Prerequisites above.
- Understand the performance implications of using different data retrieval rules types provided in Service Catalog. For more information, see Performance Considerations Before Choosing Data Retrieval Rules Types.

Step 1 Choose **Service Designer** > **Active Form Components**, and select the active form component to which the rule will apply.

Step 2 Click the **Active Form Rules** tab.and choose **New Rule** > **New Data Retrieval Rule**.

Step 3 In the first page of the Data Retrieval Rule wizard, enter a unique name and a description for the rule, and specify the Rule Type and Query Type.

a) Specify Rule Type:

- Distributing Rule: Choose this option if the primary purpose is to return the results of a query to the form—for example, in a select list, in a set of fields triggered by a selection or data entry in another field, or in a grid. You will be able to attach this type of rule to any number of form- and field-level events.
- Validating Rule: Choose this option if the primary purpose is to validate the data entered on the form against a set of results returned by a query thus ensuring that it conforms to certain standards such as security. This rule can therefore be used to prevent malicious users from intercepting the form and manipulating the content.

Note This rule can be performed on the post-Submit event (server-side) only; you will not be able to attach the rule to any other events.

- Distributing Rule with implicit validation performed on the post-Submit event: This option is checked by default when you create a new rule. Choose this option if the primary purpose is to return the results of a query to the form, but you have an additional need to validate the form data on the post-Submit event. You will be able to attach this rule to any number of form- and field-level events, and a "validating equivalent" of this rule will automatically be attached to the post-Submit event. For example, if you wish to populate a drop-down control with a list of server provisioning locations, and the locations are limited

by the form user's role, using the "Distributing with implicit validation" option means that a malicious user would not be able to specify "Seattle" when the rule excludes "Seattle" as a valid choice for his role.

b) Specify Query Type:

- **Database Table Lookup:** Database Table Lookup is the simple or an easier option. If all the data you want is available within one table (or within a database view that a Database Administrator has created), use this type. The wizard will walk you through a set of screens to define your query, all by filling in dialog boxes. Specify the data source and table in which the data reside. Once you specify the datasource, the drop-down list for the Table Name will be populated with all tables accessible in that data sources.
  - **REQUESTCENTERDS**: Contains the tables in the Service Catalog database.
  - **DATAMARTDS**: Contains the tables in the Data Mart database.
  - **Standards**: Contains the tables you created using Service Item Manager's Design Standards tab or imported using Service Item Manager's Import Data tab.
  - **Service Items**: Contains the Virtual Machine service item, the ServiceItemHistory and ServiceItemSubscription tables, and any service items defined in Service Item.
- **Enter Your Own SQL Query** : Use this query type for more complex query. If you choose this Query Type, you must specify the data source from which the information is to be retrieved, and write the complete SQL SELECT statement to be executed. To reference the value of a field on the form, the query must include the lightweight namespace that refers to that field.

```
For example: select distinct CMN_NAME from ASSET_TRACKING where MODEL = #ST_HARDWAREKIT.ComputerName#
```

Note

- The parser will automatically insert a single quote around the value returned for the namespace at runtime. This is true for all data types except "datetime." The value for a datetime data type must match exactly the datetime format expected by the RDBMS. Each RDBMS may have a different configuration for the datetime format
- If you choose “Database Table Lookup”, the generated SQL query appears at the end of the wizard in the SQL Query field (the field name changes to Generated SQL Query when saved). This gives you a starting-point from which you can then construct your own SQL query, via the second type (that is, you can copy the generated SQL query, then go back to Step 1 of the wizard, choose Enter Your Own SQL Query and paste the generated SQL query into the SQL Statement field).
- REST Web services: Using REST web service dynamic data retrieval rule, the data on the service forms can be fetched from an external system. If you choose a Query Type of **REST Web service**, you must specify the following:
  - Connection Identifier: Select the Unique Identifier or Provide Namespace from the drop-down list.
    Note If you select Provide Namespace, the text box for entering namespace appears.
  - REST URL: Specify the URL of the web resource from where the information has to be accessed using GET and POST operations.
  - REST Method: Use GET or POST method to access the web resource. When using POST method, specify more information in the form of properly formatted xml layouts, or payloads.
  - Username and Password: Basic authentication to access web resources when making a request.
  - Authentication Type: You can access the web resource using the session based and header based authentication, in addition to basic username and password-based authentication mechanism. Session-based name-and-password authentication includes additional authentication token parameter for authenticated HTTP requests.

Step 4 Click **Next**.
If you specified the Rule Type as a **Distributing Rule** or **Distributing Rule** with implicit validation performed on the post-Submit event, you need to choose one or more triggering events.

Note Always use the “Distributing Rule with implicit validation” rule type for form fields that should be strictly controlled based on entitlement or access permissions. The implicit validation prevents malicious attempts to manipulate the field values and to pass unauthorized data to the server. When applying implicit validation to a numeric field that has decimals, you may need to construct the query such that the number of decimals returned by the query will match the user input. Otherwise the validation may fail just because the number of decimal places are different.

Step 5 Select Triggering Event(s) and click Next. The rule can be executed (triggered) by checking the check boxes for the following events. See **Fields on Select Triggering Events page** table.

Note Grid fields cannot be chosen as the “Triggering Field” when creating a data retrieval rule.

Step 6 **Define Lookup Conditions** (**Where Clause**) and click **Next**.

- ◦ For a Distributing Rule or “Distributing Rule with implicit validation performed on the post-Submit event”: Rules with the retrieval type of Database Table Lookup typically need to have an associated “Where” clause as part of the criteria, which specifies which row or rows should be retrieved from the specified table. Any number of conditions may be entered by clicking “Add Where Clause”. As you enter each one, click OK. The condition just specified appears at the top of the page. If you use multiple conditions, all must be true (that is, the conditions will be grouped together using to Boolean operator AND) for a row to be returned.

- If no Lookup Condition is specified, all rows in the specified table will be returned. This typically occurs when you are populating a drop-down list (single-select or multi-select). Rather than attaching such a rule to a Form Load event, it might easier and more efficient to simply define a table-based option list as part of the field's Display Properties.

Note Grid dictionaries cannot be used to create a Lookup Condition.

Step 7 **Define Sort Conditions** and click **Next**.
If you expect the rule to return more than one row, you may want to sort the rows returned. For example, if you are using the rule to retrieve data that is being used to populate a drop-down list, the results should typically be sorted, so they are in an appropriate order. Any number of sort fields (Table Column Name) can be specified, and a Sort Direction (Ascending or Descending) specified for each. Click **Add Sort** to add a new sort field. As you enter each one, click **OK** to add the new field to the Sort Conditions displayed at the top of the page.

Step 8 **Use Lookup results on the Form**
For a Distributing Rule or "Distributing Rule with implicit validation performed on the post-Submit event", at least one distribution target must be defined. A "distribution" defines how the values returned from both table-based lookups and SQL queries are used on the form. Distributions map from column values returned in the query to fields on the service. Each rule may include one or more distributions.

In case of REST web services, services returns JSON response, therefore specify the JSON path from where values are mapped to the dictionary field.

For example, a rule used to populate a drop-down list may have just one distribution (mapping the column to a dictionary field that has the HTML representation of a single- or multi-select element. Alternatively, a rule may have multiple distributions, each mapping from one column to a dictionary field. The target dictionary fields need not be writeable on the form, they can be read-only or hidden. Any number of distributions may be entered by clicking **Add Distribution**. As you enter each one, click **OK**.

A rule cannot distribute results to a combination of grid and nongrid dictionaries, nor to two different grid dictionaries. Once you have chosen one grid dictionary field as a distribution target, any additional targets will be limited to fields in that same dictionary.

The target of the distribution may be a field on a dictionary that is not included in the current form component. It is the responsibility of the service designer to ensure that any dictionary referenced is included in another form component which, in turn, is included in a service with the current form component.

Step 9 Click **Next**.

Step 10 **Validate Field Values** and click **Next**.
If you specified the Rule Type as a Validating Rule, you need to choose one or more fields to validate.

Use this step to choose the fields that will be validated against the columns returned by your query—that is, the query results. The value of each field you specify here will be checked after the form is submitted, against the corresponding query results.If the field's value does not match any of the results, the submission will fail and the end user will receive a message to that effect. Any number of validations may be entered by clicking **Add Validation**. As you enter each one, click **OK**.

A rule cannot validate to a combination of grid and nongrid dictionaries, nor to two different grid dictionaries. Once you have chosen one grid dictionary field as a validation, any additional validations will be limited to fields in that same dictionary.

The validation field may be a field on a dictionary that is not included in the current form component. It is the responsibility of the service designer to ensure that any dictionary referenced is included in another form component which, in turn, is included in a service with the current form component.

Step 11 **Review and Save:**
The rule definition displays on the last page of the wizard. Click Save to save the rule, Cancel to discard the rule (or changes made in this session of the wizard), or Previous to return to a previous page of the wizard. The rule definition may also appear by choosing the rule on the Active Form Rules page.

Fields on Select Triggering Events page

Table 17: Fields on Select Triggering Events page

Field	Description
Form Load	When the service form is initially displayed in My Services or Service Manager
Before the form is loaded (server-side)	This server-side rule is executed on the server prior to sending it to the browser.
After the form is submitted (server-side)	Available for selection only for a Distributing Rule, this server-side rule is executed on the server after the form is submitted on the browser and sent to the server for processing.
Dictionary Field Action	In response to an event that occurs as a user is working with the form, filling out data and moving from field to field.
Dictionary Name, Dictionary Field Name, and Event	If you specify that the Event is a “Dictionary Field Action”, you must define that action by choosing the Dictionary Name, Dictionary Field Name, and Event from the drop-down menus. The “triggering” events are similar to, but not identical to, events that web page designers may be familiar with. The list of available events may vary, depending on the Input Type assigned to the field. For example, a radio button has an event “When the item is clicked”, which is not applicable to a text field.

Performance Considerations Before Choosing Data Retrieval Rules Types

Though executing a query on the server are faster than queries being called by the browser, it is recommended to execute data retrieval rules on the on-Load event rather than pre-load event. Consider the following if you plan to execute pre-load event than on-load event:

- Anything executing on the server-side (pre-load event) uses cycles on the application server vs. being confined to the user’s browser session.
- Executing rules that return number of records affects overall solution performance if executed during a server-side event. Such rules executing on the application server may affect the overall performance of

the solution, for all users. Therefore, refine your query as much as possible, to avoid returning too many results. This will result in the best end-user experience and the best performance.

- Another factor is the perceived performance to the end-user of the form. A complex service form with tens or hundreds of fields will take some time to fully load into the browser window. If that form also contains data retrieval rules that populate drop-downs on the on-Load event, the user is likely to see the form being painted first, followed by the drop-downs being populated. In between the form painting and the drop-downs being populated, the user may see what appear to be "empty" drop-down controls on the form. This effect can be mitigated if you tie the rules populating the drop-downs to the pre-Load event instead. However, moving data retrieval rules to the pre-Load event means that the form does not get loaded—and therefore the user does not see it loaded into the browser—until the rules finish executing. So although the overall performance of the complete loading of the form may be improved, the user's perception may be that the application is slow to respond to clicking the Order button.

You can try to compare the responses by tying a set of rules to the on-Load event and subsequently change them to be tied to the pre-Load event. If you make this comparison while the application is under load, you are better able to see the effects and choose the approach that is most desirable.

Modifying Active Form Rules

In order to modify an existing data retrieval or conditional rule, you must edit the rule, apply the desired changes, and navigate through all pages of the Rule Wizard. The Save button is available only on the last page of each wizard. This process ensures that all aspects of the rule are internally consistent.

Changes to the dictionaries and fields may not automatically be propagated to a rule that uses the corresponding dictionary, field or lightweight namespace. If you change the name of a dictionary or field, you must edit the rule, navigating through all pages. For conditional rules and table-based data retrieval rules, references to the dictionaries or fields are updated automatically as you proceed through each page. For SQL entry data retrieval rules, you must re-enter the SQL code, using correct lightweight namespaces.

If you delete a field or dictionary, you must edit rules to remove references to the deleted object. If a rule that previously worked suddenly stops working, a renamed or deleted object still referenced in the rule is a probable cause.

Adding Form Rules to a Service Form

The Active Form Behavior tab allows designers to attach JavaScript functions and conditional rules to events that occur within the life cycle of a service form. On this tab, you can change the order of conditional rules. JavaScript always executes at the end.

Step 1 Choose **Service Designer** > **Active Form Component**.

Step 2 Click the **Active Form Behavior** tab. The first row in the "Form or Field" column, Active Form Component, is chosen by default. The Triggering Event column lists all form-level events.

Step 3 In the Triggering Event column, choose the form-level triggering event from the available list to which the JavaScript or conditional rule will attach.

Note It is recommended to choose server-side form-level triggering rules as these are secure and less susceptible to interception attacks.

- **Before the form is loaded (server-side):** This server-side rule is executed on the server prior to sending it to the browser. See Managing Service Items on an External System for more information.

- **After the form is submitted (server-side):** This server-side rule is executed on the server after the form is submitted on the browser and sent to the server for processing. See Managing Service Items on an External System for more information.
- **When the form is submitted (browser-side):** The code or rule is executed after the Submit or Update button is clicked, and after any validations specified in the form's Display (such as checking for numeric or mandatory data), but before the form is submitted to the server.
- **When the form is loaded (browser-side):** The code or rule is executed when the service form is initially displayed in Service Catalog, Service Manager, or My Services.
- **When the form is unloaded (browser-side):** The code or rule is executed when the service form is closed.

Note JavaScript functions cannot be attached to server-side triggers.

Step 4 Click **Add JavaScript** or **Add Rules**. A dialog box lists all scripts or rules previously defined.

Step 5 Choose the functions or conditional rules previously defined by checking the checkbox next to them, and then click Add. You can use the Search box, if needed.
In the Behavior column, the functions appear below the JavaScripts section, and the rules appear below the Rules section. Review the order in which the rules are to be executed, and change the sequence if required.

Note Although you can attach multiple JavaScript functions to the same event in a single form, this practice is best avoided because the order in which the functions are specified (and executed) cannot be defined. Therefore, if you need functions to execute in a certain order, create one JavaScript function that contains all functions in the desired order.

Step 6 If required, you can edit arguments for a JavaScript function by clicking on a JavaScript function in the Behavior column, and then clicking **Edit Arguments**.

Adding Form Rules to a Service Form Field

Step 1 Choose **Service Designer** > **Active Form Component**.

Step 2 Click the **Active Form Behavior** tab.

Step 3 Click the field you want in the “Form or Field” column. The Triggering Event column lists all field-level events.

Step 4 In the Triggering Event column, choose the field-level triggering event from the available list to which the JavaScript or conditional rule will attach:

- **When the item is changed:** The user enters something into the specified form field.
- **When the item loses focus:** The user exits from the specified form field, either by tabbing to another field or clicking the mouse in another field.
- **When the item is clicked:** The user clicks on a check box, radio button, or item in a drop-down list.
- **When the item is focused on:** The user clicks in the specified field.
- **When the mouse is moved off the item:** The user moves the mouse off the specified field.
- **When the mouse is moved on the item:** The user moves the mouse onto the specified field.

Step 5 Click **Add JavaScript** or **Add Rules**.

Step 6 Choose the functions or conditional rules previously defined by checking the check box next to them, and then click **Add**. You can use the Search box, if needed.
In the Behavior column, the functions appear below the JavaScripts section, and the rules appear below the Rules section.

Step 7 Review the order in which the rules are to be executed, and change the sequence if required. If required, you can edit arguments for a JavaScript function by clicking on a JavaScript function in the Behavior column, and then clicking **Edit Arguments**.

Note Although you can attach multiple JavaScript functions to the same event in a single form, this practice is best avoided, because the order in which the functions are specified (and executed) cannot be defined. Therefore, if you need functions to execute in a certain order, create one JavaScript function that contains all functions in the desired order.

Defining Form Behavior for Service Item-based Dictionaries

Form rules and other form behavior for service item-based dictionaries (SIBDs) work almost in the same way as free-form dictionaries. Once an SIBD has been defined, it can be included in an active form component. The procedure for doing so is the same as for including any dictionary in a form component—on the Form Content tab, click **Add Dictionaries** and choose the dictionary from the popup search window. If desired, you may change the display order of dictionaries or fields in the form component.

Configuring Display Properties for Service Item-Based Dictionary

An SIBD has one unique property—the ability to automatically retrieve and prefill data about an existing service item instance. For a chosen form, this option is found on the Display Properties tab for the service-item-based dictionary.

"Enable automatic retrieval of service item instance data" should typically be checked if the service is to be used to update or delete an existing service item. To take advantage of this prefill capability, you may need two form components based on the same service item dictionary—one to be included in services where the item is created and the second to be included in services that update or delete the item.

Using Service Items in Dynamic Data Retrieval Rules

Service items are available for use in table-based dynamic data retrieval rules.

Step 1 Create the rule and specify its name, description, and triggering event.

Step 2 Choose "Database Table Lookup" as the Query Type.

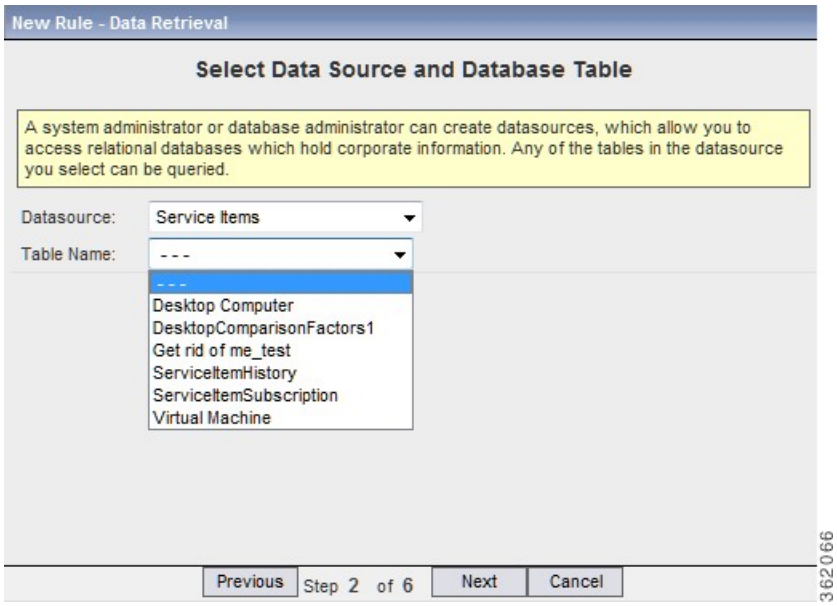
Step 3 The second page of the Rule Wizard appears. You can then choose "Service Items" as the Datasource.

The drop-down list for Table Name is populated to include:

- Any service items defined in Service Item Manager supplied with all Service Catalog installations
- ServiceItemHistory, a table automatically maintained to track the history of a service items
- ServiceItemSubscription, a table automatically maintained to track subscriptions (the current status) of service items

You can then proceed to complete the rule definition as you would for any table-based data retrieval rule. For details on defining rules, see Configuring Dynamic Form Behaviors Using Form Rules, on page 33.

Figure 3: New Rule



You can use a service item in a SQL-entry data retrieval rule by referencing the database table name which the system assigns to the service item. The table name is the prefix “Si” followed by the item name, with spaces removed. An easy way to find out the database table name (without returning to the Manage Service Items page to look up the item’s name) is to define and save a table-based rule using the service item. The generated SQL on the Summary page includes the table name.

Displaying Service Form as a Wizard

In the Service Catalog module, you can customize the display of service form with dictionaries in the form of a wizard. The wizard shows form fields in multiple pages with previous and next navigation controls. By default, all the dictionaries for a service are listed in a single page. You can configure the wizard for an individual service which helps to present the dictionaries in an organized manner, particularly if the service form contains many dictionaries.

The wizard is rendered as steps based on Active Form Components or dictionaries within the service. The wizard configured for the service has a carousel that displays all wizard steps and allows the user to navigate

to any step directly without having to page through the steps sequentially. Navigation controls and pricing information are displayed at the bottom of the page along with the action buttons that exist in service forms.

Step 1 Choose **Service Designer** > **Services** > **General**.

Step 2 For the field **Pagination View Mode**, select one of the pagination view options listed in Table 18: Pagination View Options, on page 58.

Table 18: Pagination View Options

Option	Description
Classic View	This is a default setting, i.e., no pagination is applied to the service form and contains all the dictionaries for the service.
Dictionary Pagination	Displays one wizard step per dictionary present in the service form.
Form Pagination	Displays one wizard step per Active Form Component present in the service form.
Custom Pagination	You can customize the service form to display a set of service pages with selected dictionaries associated to them. One service page with selected dictionaries is displayed as a set of wizard steps.

Step 3 Scroll down and click **Save**.

Customizing the Service Form Using Custom Pagination

To customize the service form using the **Custom Pagination** option perform the following additional steps.

Step 1 Click the icon next to the **Pagination View Mode** (this icon is available only when you select the option **Custom Pagination**).

Step 2 In the pop-up dialogue box, provide a name for the page in the **Page Name** field.

Step 3 Select the required dictionaries from the Available Dictionaries box and click the arrow to move them to the Selected Dictionaries box.

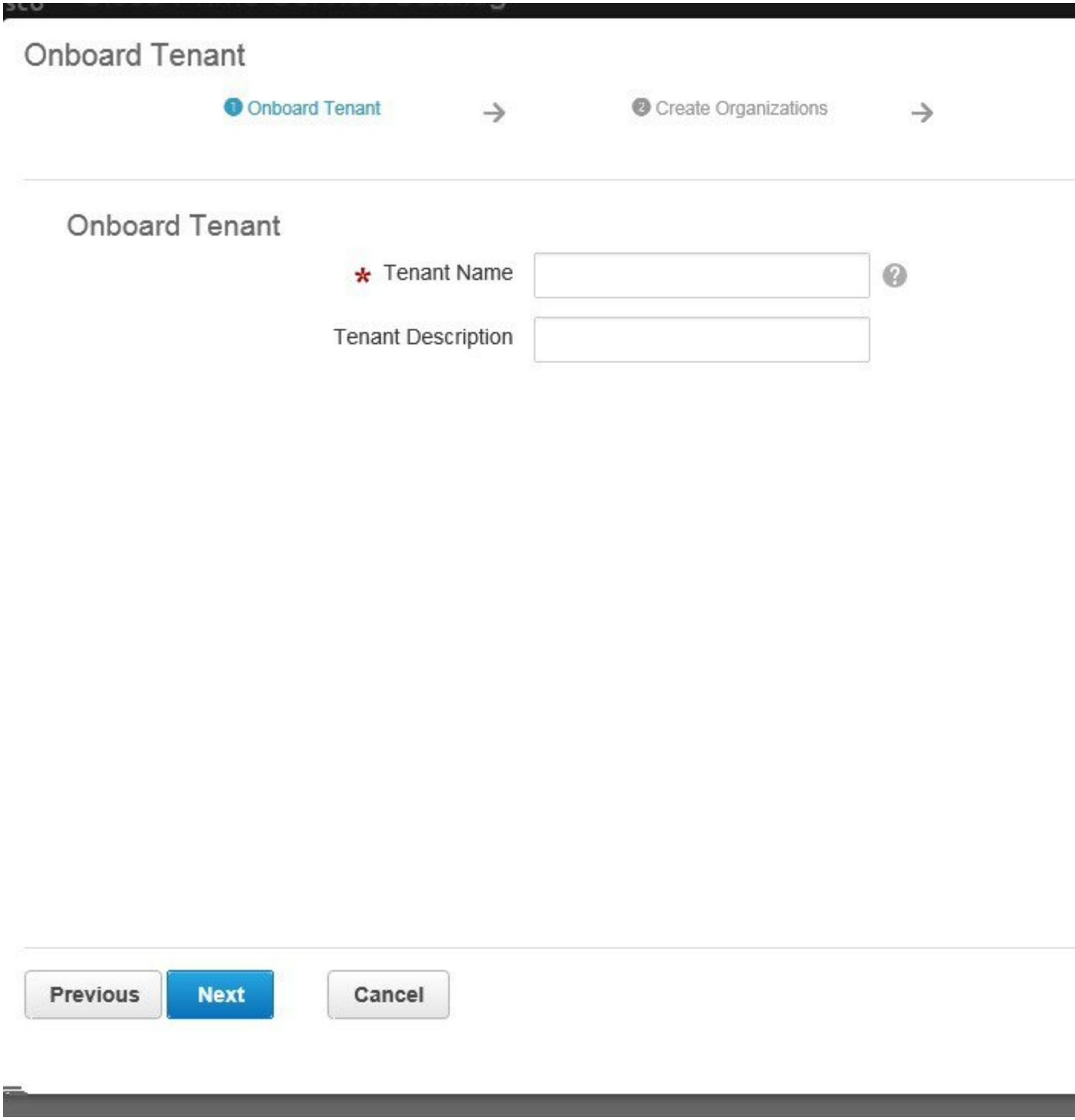
Note Ensure that the dictionaries are not repeated in the Selected Dictionaries box.

Step 4 If required, add more pages or remove selected page, using **Add New Page** and **Remove Selected**.

Step 5 Click **Save Page** and close the dialog box.

The figure below is an example of a customized service form with customized pagination for dictionaries in the Service Catalog module.

Figure 4: Sample Service Form with Custom Pagination


Onboard Tenant
1 Onboard Tenant
2 Create Organizations
Onboard Tenant
* Tenant Name
Tenant Description
Previous
Next
Cancel

Enabling Order Summary in the Service Form

The show order summary functionality allows you to add a summary page at the end of the paginated service form. This summary page is a read-only page that lists all the fields of the wizard. To enable the summary page:

Step 1 Navigate to **Service Designer > Services > General**.

Note When the Pagination View Mode option is specified other than Classic View, the Show Order Summary option is displayed.

Step 2 Check the **Show Order Summary** option.

Step 3 Scroll down and click **Save**.

Interactive Service Forms (ISF) API Overview

ISF is a set of interfaces and techniques to add JavaScript programming to a service form. JavaScript programming can only be executed when the service data is displayed—when the service is being ordered in My Services or on the Task Details tab in Service Manager.

- When Should You Use ISF and JavaScript?, on page 61
- ISF Components, on page 62

When Should You Use ISF and JavaScript?

JavaScript programming, including ISF, is meant to supplement the capabilities provided by active form rules. The most common behaviors associated with enhancing the interactivity of a service form—dynamically showing and hiding fields or dictionaries; setting field values based on the context or on data previously entered; making fields read-only or editable, mandatory or optional—can be provided by the active form rules. The server-side events (pre-Load and post-Submit) cannot, therefore, execute JavaScript.

Writing ISF and JavaScript requires technical (programming) expertise as well as the use of additional tools to debug your code, access the application server and maintain source code control. Consequently, ISF code is more expensive both to develop and maintain than equivalent active form rules. This technology should be used primarily if the desired functionality cannot be implemented via the declarative rules. Some examples are given in the following sections. These use cases typically fall into the following areas.

- Manipulating objects on the service form not accessible to the rules, such as dictionary captions, field labels, instructional (help) text, and cells in a grid dictionary.
- Performing date or numeric arithmetic—for example, computing a scheduled start date based on the date the service was ordered, or computing a service price, based on components chosen.
- Accessing commercially available, freeware-distributed or custom developed JavaScript libraries for specialized functions such as encryption or decryption, or accessing another application via a web service or server-side (AJAX) code.

Some ISF components duplicate functionality available via the active form rules. If your application's requirements can be completely met by using the form rules, that is usually the most effective way of coding. However, in order to ease maintenance of many, complex rules you might decide to implement some equivalent functionality using ISF in the following scenarios.

- Since the rules do not fully support if/then/else logic (they only support the “if” part), two rules are required for every “if” condition with an “else” clause. Once the conditions get more complicated, for example, you need nested “If statements,” the number of rules required to cover all cases would increase rather markedly. Rather than writing and trying to keep track of all those rules, it might be easier in the long run to write one ISF function, which can include nested if statements.
- Rules are bound to one particular Active Form Component (AFC), while JavaScript functions/ISF are callable from any AFC. ISF could be used for a complex piece of code that needed to be callable from many AFCs, for example, if the same dictionary was used in two different AFCs or the same piece of code needed to apply to two fields.
- If ISF needed to be performed in conjunction with some actions that could be done in rules, for example, change a field label or help text, you might consider coding the entire thing in ISF. This is because of the difficulty in ordering ISF and rules—the rules can be explicitly ordered, but the ISF must follow all the rules for the same event.

You can freely combine ISF and the declarative rules in the same form, even on the same event. The most critical limitation to be aware of in using ISF to replace or supplement actions available in the rules is that JavaScripts must be run after any rules that are attached to the same event.

ISF Components

ISF includes global identifiers as well as a set of public functions.

Global Identifiers

ISF global variables and their possible values are summarized in the table below and discussed in more detail in the following paragraphs. When these identifiers have an equivalent condition in the conditional rules, that equivalence is noted. More details may be found in the preceding sections on Active Form Components.

Table 19: ISF Global Identifiers

Global Variable	Description
Context	The module in which the service form is currently displayed. Equivalent to the “Context” condition.

Global Variable	Description
EditRequisitionBeforeOrdering	A Boolean that returns **true** in the ordering moment when an existing (previously saved) Requisition Entry is being edited and **false** under all other conditions. An alternative way to discover this condition is to evaluate (ReqID==0), since a RequisitionId is assigned when a requisition is saved. Equivalent to the condition “Does unsubmitted requisition exist?”
Moment	The current moment in the requisition life cycle. Equivalent to the “Moment” condition.
ReqCustomerID	The unique identifier of the customer for the service. It may be different from the user that is making the request. The ReqCustomerID value is available in all moments. This is a reference to the Person’s unique identifier in Service Catalog.
ReqEntryID	The Requisition Entry (also known as Service Request) ID. The ReqEntryID is zero (0) until the requisition has been saved.
ReqID	The Requisition ID (also known as Shopping Cart). The Requisition ID is zero (0) until the requisition has been saved.
ReqInitiatorID	The unique identifier of the person that initiated the service request. The ReqInitiatorID value is available in all moments. This is a reference to the Person’s unique identifier in Service Catalog.
ServiceID	The unique ID of the service; included for backwards compatibility; should not be used for services deployed via Catalog Deployer, which does not preserve entity IDs between sites.
ServiceName	The name of the service. Equivalent to the “Service Name” condition.
TaskID	The ID of the task being viewed in Service Manager. The TaskID is set for all authorization and service delivery moments. The TaskID value is zero (0) for moments in which no task is active, typically before the authorization or service delivery begins.
TaskName	The name of the task being viewed in Service Manager, with all Namespace references properly evaluated. Equivalent to the “Task Name” condition.

Global Variable	Description
UserID	The person that is making the request, or, when the context is Service Manager, the person that is working with the request.

Person References

All users must be registered in Organization Designer. The application identifies these users by assigning a unique identifier to their records in Organization Designer. The application tracks the initiator of the current requisition (ReqInitiatorID); the customer for the current requisition (ReqCustomerID); and the user currently working with the requisition (UserID).

In the ordering moment, the ReqInitiatorID is always equal to the UserID. In service delivery and authorization/review moments, the UserID identifies the task performer or reviewer. The ReqCustomerID is different than the ReqInitiatorID if the Order On Behalf (OOB) capability is used; otherwise, these values are identical.

JavaScript Functions

JavaScript functions are built into the ISF framework. ISF is an object-oriented framework. The ISF JavaScript functions are actually methods which are based on the base object serviceForm. To hide a dictionary, for instance, the user calls

```
serviceForm.DictionaryName.setVisible(false).
```

To hide a field the user calls

```
serviceForm.DictionaryName.FieldName.setVisible(false).
```

Tip In the following sections, bold and italicized typeface means that the programmer should substitute the name of the item.

Note JavaScript functions cannot be assigned to any event of grid dictionaries and their fields.

Dictionary-Level Functions

- Dictionary Permissions and ISF Dictionary-Level Functions
- Dictionary Permissions and Administrative Users
- Checking for the Existence of a Dictionary

For grid use, see the Designing Grid Dictionaries for Fields with Multiple Data Instance, on page 31.

Dictionary Permissions and ISF Dictionary-Level Functions

The appearance (and HTML representation) of a dictionary specified as read-only via Service Designer (for a specified moment or set of participants) is different from the appearance (and HTML representation) of a dictionary that is set to read-only via the ISF *dictionaryName***.setReadOnly()** function.

- When the dictionary is set to Edit only via Service Designer, no HTML input tags are generated for the fields which comprise the dictionary; they are rendered on the service form as text.
- When the dictionary's Access Control includes Edit capability and it is set to read-only via ISF or a conditional rule, the dictionary fields are displayed as input objects; however, they are not enterable.

When a dictionary is read-only at design time (that is, does not have Edit permission for the current participant and moment as specified in the Access Control tab for the Active Form Components in Service Designer) it cannot be made writeable through ISF or rules. This is true because when the read-only dictionary is rendered, the resulting HTML includes text with <span ../> tags; HTML <input ..> tags are not present.

Dictionary Permissions and Administrative Users

Dictionary permissions are ignored for a user who has been assigned the “Manage Service Dictionaries” capability. (This capability is automatically granted to any users who are in the “Site Administration” organization and may be included in user- and Service Catalog-defined roles as well.) The dictionary will appear as if it were editable. Care should be taken to not test any ISF when logged in as an administrative user, since the “Manage Service Dictionaries” capability overrides the designated dictionary permissions.

Checking for the Existence of a Dictionary

The ISF expression:

```
serviceForm.dictionaryName
```

is not a function which returns a Boolean. The *dictionaryName* is an attribute of the serviceForm object. The **dictionaryName** attribute has a value of true if the dictionary exists in the service in which the ISF is executed, or undefined if the dictionary does not exist. Therefore, robust code that checks for the existence of one or more dictionaries and takes action only if the dictionaries are present in the current service might be coded as follows:

```
AIT_Server_onLoad() { if (serviceForm.RC_CUSTCODES != undefined)     {RC_CUSTCODES_onLoad();)

  if (serviceForm.RC_PERFORMWORK != undefined)     {RC_PERFORMWORK_onLoad();)
}
```

Field-Level Functions

- Checking for the Existence of a Field
- Getting the Value of a Field
- Setting a Field to Read-Only
- Setting a Field to Visible
- Setting the Value of a Field

Functions that are applicable to fields are summarized in the table below and explained in more detail in the following paragraphs.

Table 20: Field-Level Functions

Function	Usage	Grid Use
serviceForm.**dictionaryName.fieldName**	Returns the field object if the field exists in the form, and undefined otherwise.	No
serviceForm.**dictionaryName.fieldName**.getValue()	Returns the current value of the field.	No
serviceForm.**dictionaryName.fieldName**.setValue(inputValues)	Sets the value of a field.	No
serviceForm.**dictionaryName.fieldName**.getCellValue (RowIndex)	Returns the cell value of the grid column at specified RowIndex.	Yes
serviceForm.**dictionaryName.fieldName**.setCellValue (RowIndex, inputValue)	Sets the cell value of the grid column at the specified RowIndex. inputValue takes a single value instead of an array.	Yes
serviceForm.**dictionaryName.fieldName**.setValue(inputValues, defaultValue)	Sets the value of a field. inputValues has to be an array, the first value of inputValues is assigned to any single-option field. The defaultValue is applicable only for a field with an input type of text or text area. For any other field type the defaultValue is ignored.	No
serviceForm.**dictionaryName.FieldName**.setSelection(inputValues)	Set the value of a field with an HTML representation of select (single), select (multiple), check box or radio button.	No

Function	Usage	Grid Use
serviceForm.**dictionaryName.FieldName**.setMandatory(Boolean)	Makes the field mandatory or optional. If the field is mandatory via Service Designer settings, it cannot be made nonmandatory. When a field is made mandatory, validation is automatically assigned; an error message, "Required: Please fill out this field before submitting" is displayed if the form is submitted with this field blank.	No
serviceForm.**dictionaryName.fieldName** .isMandatory()	Checks it the field has been marked mandatory. Returns true if the field is mandatory, either via the ISF setMandatory() function, a conditional rule, or a Service Designer Display setting.	No
serviceForm.**dictionaryName.fieldName** .setReadOnly(Boolean)	Makes the field or grid column read-only or read-write.	Yes
serviceForm.**dictionaryName.fieldName** .isReadOnly()	Returns true if the field or grid column is read-only and false otherwise.	Yes
serviceForm.**dictionaryName.fieldName** .setVisible(Boolean)	Makes the field or grid column hidden or visible (displayed). If the column is hidden in Service Designer settings, it cannot be made visible.	Yes
serviceForm.**dictionaryName.FieldName**.isVisible()	Returns true if the field or grid column is visible and false otherwise.	Yes

Function	Usage	Grid Use
serviceForm.**dictionaryName.FieldName**.setFocus(Boolean)	Sets focus to the field; applicable to all input types except for radio buttons and check boxes. True: focuses the field; False: blurs the field. This function does not apply for fields that are in dictionaries set read-only via Service Designer or for hidden fields—the call is ignored and no error is shown.	No
serviceForm.**dictionaryName.FieldName**.setFocusViaValidation(Boolean)	Sets focus to the field, applicable to radio buttons and check boxes only.	No
serviceForm.**dictionaryName.FieldName**.getInstructionalText(stripTags)	Returns the instructional text for a field or grid column, optionally stripping any HTML from the text based on the Boolean stripTags argument.	Yes
serviceForm.**dictionaryName.FieldName**.setInstructionalText(text)	Sets the instructional text for a field or grid column.	Yes
serviceForm.**dictionaryName.FieldName**.getPrompt(stripTags)	Returns the prompt for a field or grid column header—optionally stripping any HTML from the prompt based on Boolean stripTags argument.	Yes
serviceForm.**dictionaryName.FieldName**.setPrompt(prompt)	Sets the prompt for a field or grid column header.	Yes

Checking for the Existence of a Field

The ISF expression:

`serviceForm.dictionaryName.fieldName`

does not return a Boolean. The *fieldName* is an attribute of the serviceForm.*dictionaryName* object. The *fieldName* attribute returns undefined if the field does not exist in the current service. (The field may be hidden,

and it is still considered to exist.) Therefore, robust code that checks for the existence of one or more dictionaries and takes action only if the dictionaries are present in the current service might be coded as follows:

```
RC_REQUESTEDBY_onLoad() {  if (serviceForm.RC_REQUESTEDBY.FirstName != undefined)
{serviceForm.RC_REQUESTEDBY.FirstName.setReadOnly();)
  if (serviceForm.RC_REQUESTEDBY.LastName != undefined)
{serviceForm.RC_REQUESTEDBY.LastName.setReadOnly();)
}
```

Checking for field existence is not necessary if code has previously confirmed that the dictionary “exists”.

```
commonOnLoad() {
  if (serviceForm.RC_REQUESTEDBY != undefined) {
    RC_REQUESTEDBY_onLoad();
  }
  ...
}
RC_REQUESTEDBY_onLoad() {  serviceForm.RC_REQUESTEDBY.FirstName.setReadOnly();
  serviceForm.RC_REQUESTEDBY.LastName.setReadOnly();
}
```

Getting the Value of a Field

The getValue() method (serviceForm.*dictionaryName.fieldName* .getValue()) always returns an array. If there is only one item then it is an array of one. Use .getValue()[0] to access the first element. The getValue() method works on all fields, regardless of whether the field is read-only, read-write, or hidden in the current moment, provided that the dictionary is defined with edit access.

- For input types like check box and Select (Multi), getValue() is processed incorrectly if the values contain tabs (for example, if an attempt was made to import data from an external source where tab is used to delimit distinct values in lists). The tab character is represented within the value as \t.
- For complex controls (radio, check box, multi-select, single-select/drop-down) the value returned is the set of “selected values”—meaning only the highlighted values are returned and the “Value” property is used rather than the label or text often seen by the user.

For security reasons, this method does not work for a field with an input type of password. It returns a blank string—no error is shown.

Setting a Field to Read-Only

The setReadOnly() method (serviceForm.*dictionaryName.fieldName* .setReadOnly()) toggles a field between read-only or read-write.

- Fields with the input types of person, date or datetime have an associated button that the user clicks to choose a value for the field. Setting these fields to read-only disables the Select button next to the field so it cannot be clicked. The text field containing the descriptive information (person's name, date, or datetime) is always read-only.
- When a dictionary is marked as read-only for a particular moment in Service Designer, fields appear on the service form as boilerplate text; no HTML input object is generated. Such a dictionary (or fields in the dictionary) cannot be made read-write through ISF. Attempts via ISF setReadOnly() to make the dictionary or a field in the dictionary writeable silently fail. Similarly, attempts via ISF to make the field or dictionary read-only have no effect and do not generate an error.
- If a dictionary or field is made read-only through ISF, the HTML input box is still displayed but field contents cannot be edited and the field is removed from the tab sequence.

Setting a Field to Visible

The setVisible() method (serviceForm.*dictionaryName.fieldName* .setVisible()) toggles a field between being hidden and visible on the service form.

- When a dictionary is marked as hidden for a particular moment in Service Designer, fields in that dictionary cannot be made visible via ISF.
- When a dictionary has been hidden via ISF, attempts to make fields in the dictionary visible silently fail.

Setting the Value of a Field

Two methods are available for setting the value of a field:

- serviceForm.*dictionaryName.fieldName* .setValue())
- serviceForm.**dictionaryName.fieldName**.setSelection())

A third method is the equivalent of the setValue() method, but for cells in a grid:

- serviceForm.dictionaryName.fieldName.setCellValue())

See the Configuring Dynamic Form Behaviors Using Form Rules, on page 33 for more details.

The setValue() method sets the value of the specified field to the specified inputValues. inputValues has to be an array; the first value of inputValues is assigned to any single-option field.

- For security reasons, this method does not work for fields with an input type of password.
- inputValues for Select (single), Select (multiple), check box and radio button input types are set to the display values and this function selects the same. Check box type fields can take an array of inputValues. The function returns without selecting when invalid display values are passed that are not in the field. On saving the service form, the selection is persisted.
- No validation is performed on the inputValues. It is possible for wrong values set through this function to be persisted when the service form is submitted, even for field types like Person, SSN, URL, or Date.
- For Person type fields, see the Specialized Field-Level Functions, on page 70 section below.

The setSelection() method should be used for fields with multiple options—select (single), select (multiple), check box, and radio button. The argument is matched to the values for the various elements in the field. For a radio field or check boxes, this function allows you to set the selection to one or none of the controls. For example: .setSelection([‘’]) clears all the radio buttons and restores the “pristine state”.

For a multi-select or a single-select input item, setSelection marks the requested items selected. Items that are not found in the list are ignored.

This function does not do anything for fields with an input type of password, text, or textarea, or for fields in read-only dictionaries.

Specialized Field-Level Functions

- Person-Based Fields
- Fields allowing Multiple Values/Selections

Some field-level functions are applicable only to fields of particular types. These are summarized in the table below.

Table 21: Special Field-Level Functions

Function	Usage
serviceForm.*dictionaryName.fieldName_disp* .getValue()	Applies to Person type fields only. fieldName.getValue(inputValues) gets the PersonID of the specified field. The display value of the Person field (generally the person's name) can be accessed by suffixing '_disp' to the fieldname: ...fieldName_disp.getValue().
serviceForm.*dictionaryName.fieldName_disp* .setValue(inputValues)	Applies to Person type fields only. *fieldName* .setValue(inputValues) sets the PersonID of the specified field. Use *fieldName_disp* .setValue(inputValues) to set the value displayed in the text box.
serviceForm.*dictionaryName.fieldName_saved* .getValue()	For single- and multiple-select fields, the normal fieldName.getValue() always returns the value that is currently selected. The saved values for Select type of fields can be accessed by suffixing "_saved" to the actual field name –. *fieldName_saved* .getValue() This function can be useful to determine the values that was saved in previous moments for Select lists. This function also returns values previously saved in the same moment. Although *fieldName_saved* .setValue() executes without error, it is nonfunctional, and does not change the saved value of the field. **Note** This function is not available for grid dictionaries.

Person-Based Fields

The Person Data type and HTML representation are designed for the display and validation of person profile data stored in Organization Designer. The field appears on the service form as a single-line text box with an associated control labeled “Select”, as shown below.

Figure 5: Select Option

Clicking the “Select” button opens a popup window allowing the user to search for people in the Service Community, and choose the person of interest. The person’s name and email address (fields configured in Site Administration) are displayed on the service form.

ISF functions operate as follows when applied to a person-type field.

- *personFieldName* .getValue() returns the ID of the person, the unique identifier for the person record in Service Catalog. That ID can be used, with appropriate, customized, server-side code to lookup additional person profile information for display in other fields on the service form.
- *personFieldName_disp* .getValue() returns the name of the person as currently displayed on the service form.
- Using the Select control automatically updates both the PersonField and PersonField_disp fields, so they are kept in sync.
- *personFieldName* .setValue() may be used to set the value of the PersonID. This function can be used in conjunction with *personFieldName_disp* .setValue(), so that the correct name is always displayed for the current ID.

The PersonField.setValue () function expects a valid Person ID, but no validation is done by the client—so assigning an invalid ID does not cause the submit/update action to fail. The display value of the PersonField can be accessed by suffixing ‘_disp’ to the fieldname:

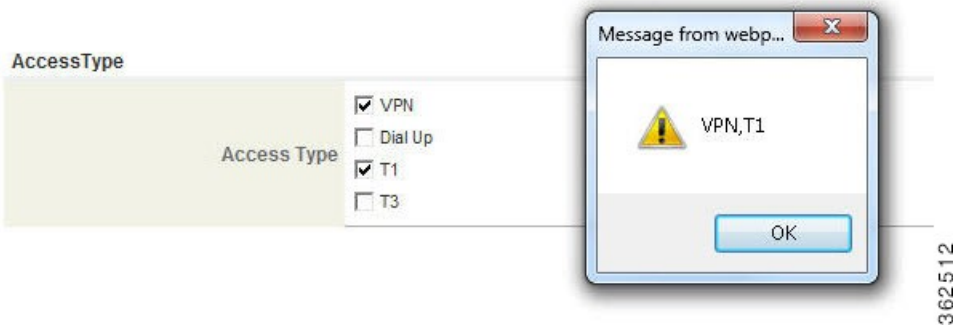
fieldName_disp .setValue(inputValues).

- When the service form is submitted, the Person ID value that was set using this function is retained—the display value is ignored.
- When the Person name changes, the service form does not synchronize the Person display value; that is, it stills show the value that was set using this function.

Fields allowing Multiple Values/Selections

The HTML displays for multi-select fields and for check boxes allows multiple values to be chosen for the same field. The values chosen are represented as a comma-separated list of values, as shown in the following example:

```
alert (serviceForm.EUIT_RemoteAccessDetails.AccessType.getValue([0]));
```



A JavaScript split() method can be used to parse the field value into its distinct elements.

Integrating ISF Code into Service Forms

ISF and service forms implement an event model that is similar, but not identical, to the Document Object Model (DOM) event model. That is, customized JavaScript functions may be invoked to handle events that occur during the processing of a service form.

JavaScript functions are typically invoked as event handlers for processing events that occur as an HTML form is displayed and the user enters data in the form's input fields. Since service forms are generated dynamically, based on the dictionary and form definitions previously stored in the repository, it is not possible for programmers to simply type ISF code into an HTML file. They must rely on the user interface provided by Service Designer to write their functions and to attach these functions to the appropriate event. Therefore, any JavaScript function to be accessed as an event handler must also be defined within the repository. Such functions are defined via the Scripts option of Service Designer, and associated with the appropriate event via the Active Behavior tab for the form.

JavaScript functions written as event handlers can, in turn, call other JavaScript functions. These functions (if they are to have a public scope) cannot be defined as Scripts within Service Designer. Instead, they must be written in a JavaScript library, a text file comprising one or more functions, which resides on a file system accessible to the application server. The Service Designer interface is then used to include these libraries in forms in which their functions are required.

Global Form Settings

Use the global form settings to manage the behavior of JavaScript that is used in all forms. These settings comprise the standard form events:

- When the form is submitted (browser-side)
- When the form is loaded (browser-side)
- When the form is unloaded (browser-side)

Note The server-side triggering events, “After the form is submitted (server-side)” and “Before the form is loaded (server-side)”, are only available for form rules.

Adding JavaScripts

- The General tab allows you to create and maintain a JavaScript function.
- The Libraries tab allows you to include a JavaScript library in the current function and, by extension, in the form to which the function is attached.
- The Active Form Components tab displays the forms to which the current function is attached.

To create a new JavaScript function:

Step 1 Choose **Service Designer** > **Scripts**.

Step 2 Choose **New** > **New Function**. Once the function has been created, choose it for maintenance from the tree structure on the left.

- **Name**: The name of the function. Although this name is just used within Service Designer, to refer to the JavaScript function, it is best practice to use a JavaScript identifier, which will identify the function within the generated code. As a JavaScript identifier, the name is a single, case-sensitive word consisting of letters and numbers and starting with a letter. For guidelines on naming functions, see the Naming Convention in Guidelines for Designing Optimal Service Forms, on page 89.
- **Description**: Optional, but highly recommended description of the function.
- **Add this script to the following events on all forms**: These check boxes provide a way to specify the function as the event handler for the checked event. This "global" attachment would replace the "local" attachment of the function to the event via the Active Form Behavior tab. Global attachment is not recommended for most projects, as discussed in the ISF Coding and Best Practices, on page 84.
- **JavaScript Function**: The actual code for the function, which includes the ISF code. The function signature must not include the "function" keyword (this is automatically added when the function is included in the generated service form), but function contents otherwise follow standard JavaScript rules. For example:

Write this code block in the JavaScript Function	To generate this code in the form:
`CER_Name_onChange() {` `}`	`function CER_Name_onChange() {` `}`

Step 3 Click **Add new JavaScript**.

Adding Arguments to a JavaScript Function

You can add arguments to JavaScript functions (every time the function is called). Function arguments created in Scripts can be overridden by the service designer at the service level.

Step 1 Choose **Service Designer** > **Scripts**.

Step 2 Select a function and choose **General** > **Function Arguments**.

a) In the Argument Name field, enter a name for the argument (follow naming standards for JavaScript identifiers).

b) In the Default Value field, enter a default value.

Step 3 Click **Add** to add the new function argument.

Step 4 (Optional) Enter a description by clicking the Information icon button . A Parameter Description popup window appears where you can enter a description and then click **Set Description**.

Step 5 **In the bottom left of the window,** click **Save**.

Follow the guidelines below in defining arguments:

- Set an unused default value (that is, some default value that will never be overridden) for each of the JavaScript arguments.
  - For example, MyUniqueFunction(arg1, arg2) { .. }: The default value for these two arguments should be - arg1 = 'unused value 1', arg2 = 'unused value 2'.

Tip You must enter a default value for each argument. Otherwise, you will encounter errors.

Tip You must enclose default values intended to be strings in single quotes. Do not use double quotes and do not use two single quotes with no value in between.

- There is no way to mark an argument as a particular data type because JavaScript is type-less.
  - If the intent of an argument is a string value then the default dummy value (the value which will never be possible) can be enclosed in single quotes. For example: 'AAABBBCCCDDD'.
  - If the intent of an argument is a number value then the default dummy value (the value which will never be possible) can be some negative value. For example: –999999999.
- Edit the function arguments after adding the JavaScript function into an Active Form Component.

Associating Libraries with JavaScript Functions

Storing JavaScript in an external JavaScript (library) file has the following benefits:

- Code for many functions is maintained in one place rather than requiring the user to navigate to different screens, as would be the case if each event had a function attached in Scripts.
- Maintaining the code in a file makes it easy to version control the source code.
- Maintaining the code in a text file makes it easy to do global search-replace an execute searches on the file.
- A JavaScript library is simply an ASCII file which resides on the application server. As such, you can use a powerful editor to maintain file contents.
- Since the same piece of code is referenced from one or more functions, changes need to be made only at one location and tested only once.

Prerequisites for Using Libraries

The library approach has the following prerequisites:

- The JavaScript libraries must reside on the web deployment directory (RequestCenter.war) on the application server. By convention the libraries are placed in a directory named “isfcode”.
- The programmer must therefore have access to the file system of the application server. This may entail creating additional logins on that server or providing additional client software (for example, a Remote Desktop service).
- The directory on which the ISF code resides needs to be set with read-write permissions for the ISF programmers.

A decision regarding where to store ISF scripts must be made before detailed design is attempted, as it will affect the design cost. It is significantly more efficient to use the library approach than to embed all ISF in the repository.

If the function calls additional functions which reside in a library, use the Libraries tab of the JavaScripts option to specify the library.

To add a new library:

Step 1 Choose **Service Designer** > **Scripts**.

Step 2 Select a function and choose **Libraries** > **Add Libraries**.

Step 3 In the Add Libraries window, select the check boxes corresponding to the libraries you want to add and click **Add** to include the chosen items in the JavaScript function.
All libraries included in the function will appear on the Libraries page.

You may delete one or more by checking the corresponding check box and clicking **Remove**.

Reviewing Forms With JavaScript

The Active Form Components tab lists the forms to which the current JavaScript has been attached.

To review the forms used for a Javascript

Step 1 Choose **Service Designer** > **Scripts**.

Step 2 Select a function and choose **Active Form Components**.

Step 3 Click on the form name to review the form definition.
This cross-reference reflects only “local” attachment of functions. It does not include any JavaScripts which have been “globally” attached to a service by checking the “Add this script to the following events on all forms” check box on the General tab of the JavaScript function.

Creating JavaScript Libraries

Use this procedure to assosciate library which you have created (or will create) external to Service Catalog.

Procedure

Step 1 Choose **Service Designer** > **Scripts**.

Step 2 Choose **New** > **New Library** to create an entry for your library in the repository.

- **Name**: Any name may be specified for the library, since this is a descriptive field. This name is used within Service Designer to refer to the library.
- **Import from URL**: The location of the library. The library must be located within the web deployment directory (designated as /RequestCenter/). By convention, ISF libraries are placed on the isfcode directory, directly beneath the root directory /RequestCenter.
- **Include this library in all functions**: This, too, is a slight misnomer, as the check box actually includes the library in all service forms, rather than in all “functions”. Checking this check box includes the library (by reference) in the service form, so that any functions defined in the library may be invoked. (A <script> tag for the library is generated into the service form.) Details on this and other options are discussed in the ISF Coding and Best Practices, on page 84.

Step 3 Add the library to the repository by clicking **Add new Library**.

Adding Functions Arguments to JavaScript

To add JavaScript functions to a form:

Step 1 Choose **Service Designer** > **Active Form Component**.

Step 2 Click the **Active Form Behavior** tab.

Step 3 Click the first row in the "Form or Field" column, **This Active Form Component**. The Triggering Event column will list all form-level events.

Step 4 Choose the form-level triggering event to which the JavaScript functions are to be attached.
Available form-level triggering events for JavaScript functions are:

Table 22: Form-Level Triggering Events

Form-Level Triggering Events	Description
When the form is submitted (browser-side)	The code is executed **after** the Submit or Update button is clicked, and **after** any validations specified in the form's Display (such as checking for numeric or mandatory data). These actions are executed **before** the form is submitted to the server. This usually provides for a window to do extra validation, formatting, or any kind of processing before the data is sent to the server. The onSubmit function **must return a Boolean**, true if the submission can proceed, and false to stop it. This corresponds to the HTML event: form ... onSubmit.
When the form is loaded (browser-side)	This is the **preferred** place to add code that populates fields in dictionaries, or that does some kind of form massaging before the form is rendered on the browser. This corresponds to the HTML event: body ... onLoad.
When the form is unloaded (browser-side)	This event is called when the form is being closed. This event can be used to notify the user about data being lost if the window is closed. This corresponds to the HTML event: body ... onUnload.

Note The server-side triggering events, "After the form is submitted (server-side)" and "Before the form is loaded (server-side)", are only available for form rules.

Step 5 Click **Add JavaScript**.
An Add Functions dialog box appears listing JavaScript functions previously defined with the Scripts option.

Step 6 Choose the JavaScript functions you want by checking their check boxes, and then click **Add**. You can use the Search box to search for JavaScript functions, if needed.

The functions appear on the Behavior column below the JavaScripts section.

Note Although you can attach multiple JavaScript functions to the same event in a single form, this practice is best avoided, because the order in which the functions are specified (and executed) cannot be defined. Therefore, if you need functions to execute in a certain order, create one JavaScript function that contains all functions in the desired order.

Adding JavaScript Function to a Field-Level Event

To add a JavaScript function to a field-level event:

Step 1 Edit the form in the Active Form Components component of Service Designer.

Step 2 Click the **Active Form Behavior** tab.

Step 3 In the "Form and Field" column, expand the dictionary node and choose the field to which the function is to be attached.

Step 4 In the Triggering Event column, choose the field-level triggering event that corresponds to the timing at which the function is to be executed.

Figure 6: Triggering Event

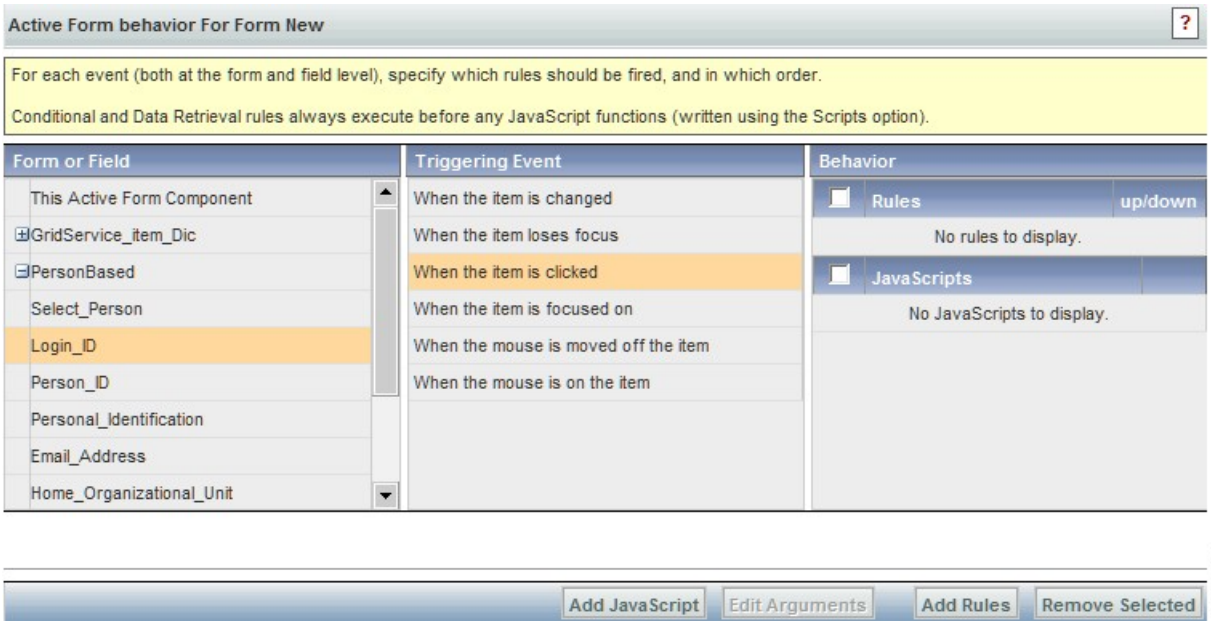


Table 23: Available field-level triggering events are:

Triggering Event Description	HTML Event Name
When the item is clicked	onClick
When the item is changed	onChange

Triggering Event Description	HTML Event Name
When the item is focused on	onFocus
When the item loses focus	onBlur
When the mouse is on the item	onMouseOut
When the mouse is moved off the item	onMouseOver

The **onChange** event is best used to detect changes to Text/Single-select fields. The onChange event for a Person, Date or DateTime field is always triggered, even if the same Person is chosen from the "Select Person" popup window, or the same Date is chosen from the Calendar popup window. Similarly, any typing in a Text field triggers the onChange event, whether the value in the field is actually changed or not.

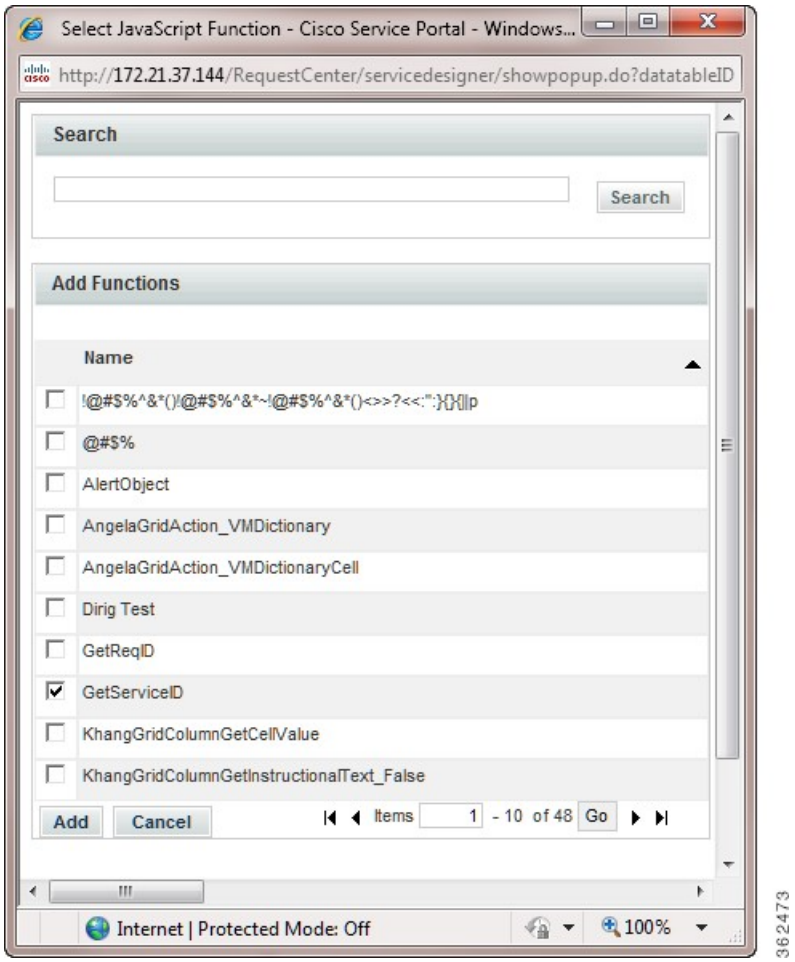
- Fill a field based on the value of the current field (for example, set a flag in a service based on the value of another field entered by the user).
- Choosing "Other" from a drop-down requires displaying an extra text box on the form.

The **onClick** event works best with Radio Button/Check Box controls. The onClick event does not trigger for Person, Date and DateTime fields as text boxes for these field types are always read-only and selection is only possible through the popup window.

Step 5 Click **Add JavaScript**.
An Add Functions dialog box appears listing JavaScript functions previously defined with the Scripts option.

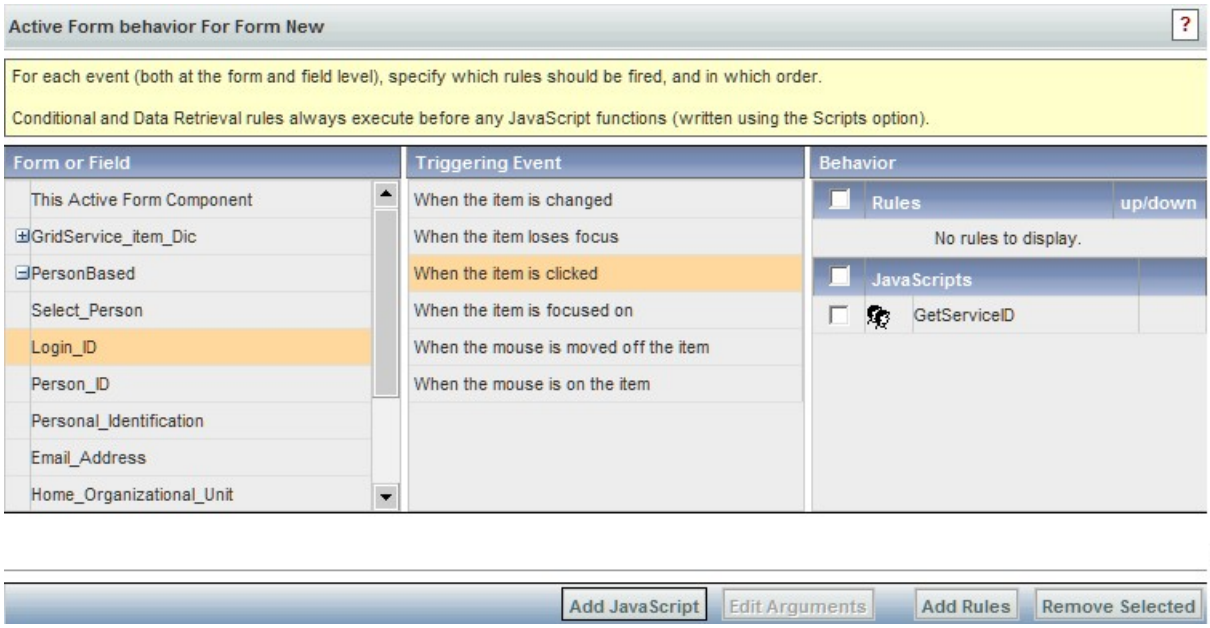
Step 6 Choose the JavaScript function you want by checking its check box, and then click **Add**. You can use the Search box to search for JavaScript functions, if needed.

Figure 7: JavaScript function



The function appears in the Behavior column below the JavaScripts section, as shown below.

Figure 8: Function in JavaScripts Function



Note Although you can attach multiple JavaScript functions to the same event in a single form, this practice is best avoided, because the order in which the functions are specified (and executed) cannot be defined. Therefore, if you need functions to execute in a certain order, create one JavaScript function that contains all functions in the desired order.

Step 7 If required, you can edit arguments for a function by clicking on a JavaScript function in the Behavior column and then clicking **Edit Arguments**. (See the Adding Arguments to a JavaScript Function, on page 75 for more information.)

Using JavaScript

This section describes the usage of Javascript function.

Associated Controls (Buttons and Links)

Any dictionary field may have an associated button, as specified in the “Add a Button” section of the Display Properties for the field:

Figure 9: HTML Representation

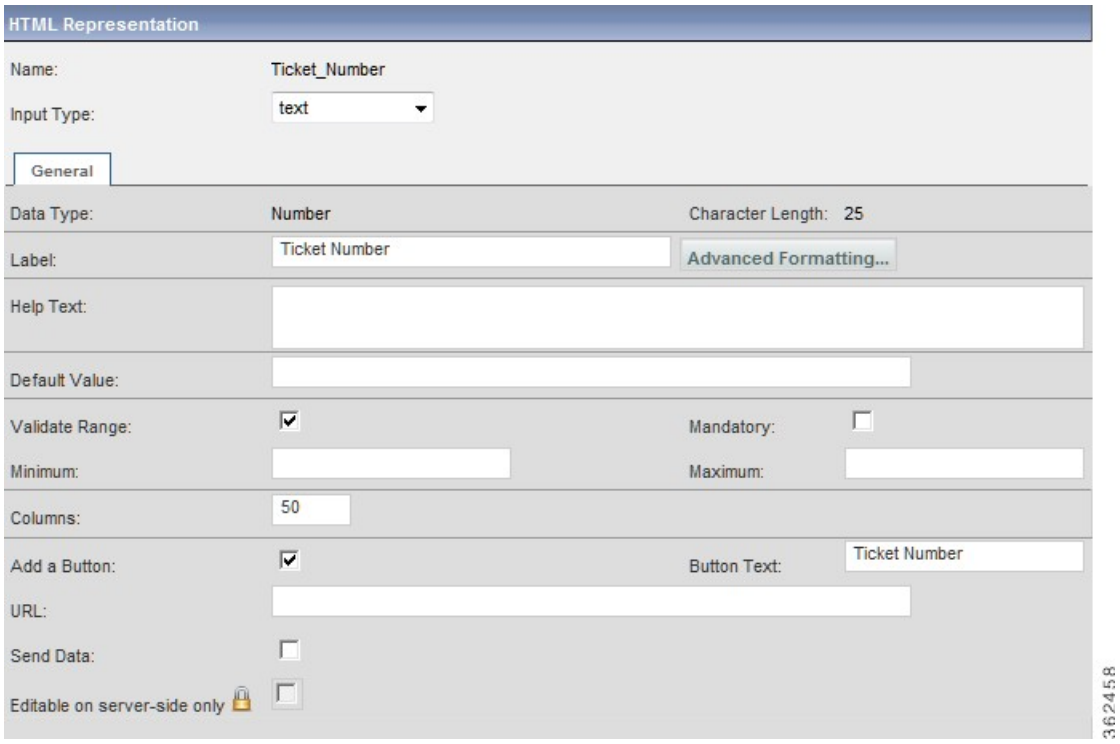


Figure 10: Remedy Ticket Numbers

You can specify a caption (text to appear on the button) and a URL. The URL can point to a JavaScript function or to an external page accessible elsewhere on the network.

When referring to an external web page, use the fully qualified URL.

- The URL may point to a JavaScript function, available in a library, or embed JavaScript code. Form data cannot be included in the parameters, if any, included in a function call. However, the function can include ISF.
- The Send Data option is applicable only to an external web page. This will POST a response to that page including all of the data on the current form.

The disadvantage of using an associated button, rather than placing code in an event handler, is that the user must use an extra keystroke (clicking on the button) to invoke the code. Consequently, buttons are best reserved for events that need to occur on demand, not in the course of regular form processing, or for extended dialog boxes or browsing of external sites.

Simply specifying a URL brings up the specified web page in a separate window. However, the window is not resizable and does not include scroll bars. Therefore, for some applications, it might be preferable to use JavaScript to display the web page in a window that you explicitly specify. Sample code is shown below.

```
javascript:window.open ("http://www.coder.com", "mywindow","status=1,toolbar=1, scrollbars=1,
 width=500, resizable=1");
```

The JavaScript all has to be on one line (no carriage returns). See sample output (primitive, but you get the idea) below. The above line is copied and pasted from the URL entry for the button with the “URL with JavaScript” label.

ISF Coding and Best Practices

Using ISF adds an additional level of complexity to a service catalog project. This section outlines some methodological and technical tips that may be helpful in developing in this environment.

- Install additional tools on your client workstation or reconfigure software previously installed for developing, testing, and debugging JavaScript and ISF efficiently.
- JavaScript does not warn of errors. Therefore, Install a JavaScript DebuggerJavaScript to have more helpful error messages and debugging facilities The Microsoft Script Debugger is available as a free download from Microsoft (find it via a an Internet search). Some web development environments, such as Visual Studio, also include debuggers, which can be used.

In order to use the debugger, you must reconfigure Internet Explorer to enable script debugging in **Tools > Internet Options > Advanced** option.

- Use a text editor that provides syntax highlighting for JavaScript programs. Several such editors are available as freeware or trial versions. Some Java integrated development environments (IDE) also offer support for editing JavaScript files.
- Provide ISF developers read-write access to the directory on the application server (typically, the isfcode directory underneath the RequestCenter.war web archive) on which JavaScript libraries reside. They will also need software for transferring the files between their workstation and the application server.
- Library files are served from the application server. Therefore, page caching must be disabled, to allow revised versions of the libraries to be loaded.

Architecture/Storing ISF Scripts

Service Designer allows JavaScript functions to be stored in Scripts (within the repository) or Libraries (as external files on the file system).

Advantages of Using Libraries

Storing JavaScript in an external JavaScript (library) file has the following benefits:

- Code for many functions is maintained in one place rather than requiring the user to navigate to different screens, as would be the case if each event had a function attached in Script Manager.

- Maintaining the code in a file makes it easier to version control the source code.
- Maintaining the code in a text file makes it easier to do global search-replace and execute searches on the file.
- A JavaScript library is simply an ASCII file which resides on the application server. As such, you can use a powerful editor to maintain file contents.
- Since the same piece of code is referenced from one or more functions, changes only need to be made at one location and tested only once.

Structuring and Using Libraries

In principle, all client code could be included in one library. However, under some circumstances it might be advisable or required to divide the code into multiple libraries.

- One or more JavaScript libraries can be used, grouping the ISF functions so that those that are likely to be used in the same service or group of services are in one library, and those used by another set of services are grouped in another. This allows multiple sets of developers, potentially working on independent projects, to keep their work separate.
- If using multiple, independent libraries, the libraries should never be attached globally to the service forms (using the “Include this library in all functions” check box on the “Library” page of the Scripts option). Instead, the relevant library (libraries) should be included in a function attached to the “onLoad event” of a form (using the “Libraries” tab of the “Scripts” page).
- Similarly, the onLoad event to which the relevant library is attached should not be included in all forms (using the “Add this script...when the form is loaded (browser-side)” option on the Scripts page). Instead, it must be associated with the event using the Active Form Behavior tab for the form in the Active Form Components.
- There is a theoretical advantage in placing ISF functions likely to be used in the same service in one library and those functions that are rarely used, or used under well-defined circumstances, in another. The advantage would be a reduction of memory used (functions not required for a particular service are not loaded). However, since memory is relatively cheap and plentiful these days, and the amount used by ISF functions is minor compared to that used by other components, so no practical advantage have been observed.

Recommended Naming and Coding Standards

If your organization has developed any JavaScript coding standards, you should use a naming and a formatting standard. Although scripts written to support ISF are not generally too long or complex, applying naming and formatting standards helps programmers understand and read one another's code.

JavaScript is case-sensitive so any naming standard needs to specify case to be used for object names.

ISF Function Names

The ISF you write will be in JavaScript functions. If the functions apply to a specific dictionary or field within that dictionary, the function names should reflect this hierarchy. This makes it much easier to track function usage. Function names follow the convention:

- dictionaryName_event

- dictionaryName_fieldName_event

For example,

- RC_REQUESTEDBY_onLoad
- ST_HighProfile_onSubmit
- ST_UserLocation_FieldOffice_onChange
- SVC_Phone_PhoneType_onClick

In addition to dictionary- and field-specific functions, Cisco Advanced Services may create the following site-wide functions, which may be modified as required:

- rc_CommonService_onLoad
- rc_CommonService_onSubmit
- siteRC_REQUESTEDBY_onLoad
- siteRC_REQUESTEDFOR_onLoad

This naming convention makes it easier to understand how each script is used, and to find the appropriate place to create any new functionality. It also allows site-specific code to interface correctly with ISF code that may have been installed in conjunction with a standard service library.

Code Placement

We recommend that ISF JavaScript functions be placed in an external JavaScript library. It is important to order the functions within that file, to facilitate finding a particular function.

Code Formatting

All code should be stored in ANSI text files without tabs, using 2-space indentation for clarity of reading and understanding. Lines should not be longer than 76 characters each.

ISF-Specific Best Practices

Be very careful changing the names of fields in a dictionary. If a field is referred to in a function, that function will stop working.

Every standard property and method in JavaScript starts with a lower case character, and the next word starts with an upper case character. For instance the property "**readOnly**" or the method "**onSubmit**" follow this convention. Although it is not enforced in ISF, following similar conventions is recommended.

Authoring JavaScripts

Service forms are generated automatically based on the specifications you enter interactively in Service Designer. But HTML tags for including libraries in a particular page or pages, or assigning a piece of JavaScript to a particular event embedded in a page are manually. You use Scripts to specify how to include libraries in the generated HTML pages that contain service forms, and the Active Form Behavior tab to instruct the application about which Scripts are event handlers for particular events. You must work outside of Service Catalog to maintain code within the libraries.

Creating a Library

Using the Libraries option under **Service Designer > Scripts**, you can place custom ISF code in one or more JavaScript libraries. Since the library is a text file, external to the application, you need a text editor to maintain the library contents. A JavaScript-aware editor or development environment is highly recommended. Third-party JavaScript libraries may also be used in conjunction with ISF.

Copying the Library to the Application Server

To be accessible by the service form, the library must reside within the directory structure on the application server, mapped to the URL/RequestCenter. By convention, the library is placed on a directory named, isfcode located directly under the root. The physical location of the /RequestCenter site may vary according to how Service Catalog is installed on your server and the application server in use. Please consult your system administrator to determine the physical location on which the ISF libraries need to reside. You also need to ensure that ISF developers have read/write access to this directory and a tool for transferring files to the directory.

Including the Library in Service Forms

A reference to the library (implemented as an HTML script tag) must be included in the generated service form in order for functions in the library to be used in the service form. This reference may be generated in one of two ways.

- Use the Libraries tab of the JavaScripts node of the Scripts option to associate the library with a particular function. The library is then available in all forms that use the function.
- Use the "Include this library in all functions" check box on the Library option of Scripts to include the library in all service forms.

Loading the Library by Including it in a Function

This is the recommended approach to including a library reference in a service form. There should be a function that is invoked as an onLoad event. Libraries are then included in that function. One or more onLoad functions may be coded. Each may have a different set of libraries attached. In this way different teams of developers have control over which libraries are available to their service forms.

Note The loading sequence of the JavaScript libraries is not controlled by the application and can be influenced by known and unknown environmental variables such as Application Server, OS version, and Database Type.

Loading the Library via the Library Check Box

The "Include this library in all function" check box on the Library page is a misnomer; it should be "Include this library in all forms" since the library is only loaded once per form. Using this check box puts the <script> tag for the library at the beginning of the generated HTML page, ensuring that the library and its functions are present when the page-level ISF calls it.

This methodology is not recommended for complex projects.

Note The loading sequence of the JavaScript libraries is not controlled by the application and can be influenced by known and unknown environmental variables such as Application Server, OS version, and Database Type.

Verifying

At this point you have a library (which possibly contains no code) and a specification to use the library for all service forms. You can verify your work by running a service form, looking at the source code, and searching for the name of your library. To view the source code of a service form, you cannot use the "View Source" option of your browser. Because the service form is displayed as a frame within an HTML page, its generated source is not displayed. Instead, move the mouse pointer to within the service form (not within a field) and use the right mouse button option to "View Source". A search shows the name of the library.

Writing Custom JavaScript Functions

Due to the difficulty of debugging in an HTML and JavaScript environment, it is recommended that your write, debug, and test one function at a time while writing the code.You may, of course, edit the library file containing your JavaScript locally; but to test it, it must be copied back to the designated directory on the application server. When initially developing code, or if your access to the application server is limited, it may be easier to write the code within a Script and refactor when you are done, moving the function to the library and leaving only a wrapper function, which calls the library function, under the Scripts. Make sure that you save often and keep backup copies in case you need to revert to a previous version of the code.

Attaching the Function to the Appropriate Events

Once the code is written, it needs to be attached to an event in the form. Since the service form HTML page is generated dynamically, you must use Service Designer to do this. See the Adding Functions Arguments to JavaScript, on page 78 for more information. To attach a field-level event to a dictionary field:

- Write the function in the JavaScript library. Name it *dictionaryName_fieldName_event* , for example, RC_REQUESTORLOCATION_LocationName_OnChange.
- Name the function with a site-specific code prefixed to the name of the function you previously included in the JavaScript library. Service Catalog-provided functions (for example, those used in services included in the Service Catalog libraries) use an "rc" prefix, so the code would appear as shown here:

```
rcRC_REQUESTORLOCATION_LocationName_onChange () {
 RC_REQUESTORLOCATION_LocationName_onChange();
}
```

- Use the Active Form Behavior tab of the Active Form Components option to attach the function to the appropriate field-level event. Choose the field and triggering event, then add the function by clicking Add JavaScript. (If the dictionary is used in multiple forms, the function must be attached to the dictionary field in every form. This is not recommended.)

Testing

ISF code is attached to an active form which can be reused in many services. However, it is not sufficient to test just one service. For example, you may have code that assumes that a field in a dictionary in another form

is also present in the service; if that is not the case, your ISF code will fail with a JavaScript error. Therefore, you need to set up a testing matrix based on the different combinations of forms that can be used. Especially in the initial phases of testing, it is useful to run multiple sessions of Service Catalog simultaneously.

- Start Service Designer in one browser, with the option set to Scripts (to make changes to function code) or Active Form Components: Active Form Behavior (to change JavaScript attachments) displayed.
- Run My Services or Service Manager in the other browser, to test the service form in the moments for which rules or ISF have been defined.
- If you are testing code that resides in a library you will, of course need another window: to edit the library file and upload your changes to the application server.

If a JavaScript error is encountered, the JavaScript debugger is displayed. After you have fixed the error (by editing the function or library code) and dismissed the debugger, there is no reason to exit the current service form and start a new request. Simply refresh the page to cause the current service form to be reloaded with the new ISF code.

If the Browser Cache setting is enabled in the Administration Settings, changes made to the JavaScript libraries will not take effect until the browser cache has been deleted. Therefore in a development environment, it is best to disable browser caching. When modifications to JavaScript libraries are deployed to the production environment where browser caching might be enabled, application users will need to delete their browser cache. To prompt the application users to do so, follow the instructions in the Cisco Prime Service Catalog Administration and Operations Guide to increment the browser cache version.

Guidelines for Designing Optimal Service Forms

Service Catalog enables you to create service forms quickly and easily. Following are some general guidelines and principles for designing optimal service forms.

- **JavaScript code should be specific to a dictionary**. Ensure each function in the code is stand-alone and does not depend on any other function or dictionary. This is to make sure the code functions even if a new dictionary is added or existing dictionaries are removed. For example, assume that two dictionaries used in a particular service have code that must be executed in an onLoad event. Rather than writing one monolithic function, write two dictionary-specific functions, place them in a library, and call them from a wrapper function, which is defined as a Script and attached as the onLoad event to the form:

```
Common_Service_onLoad () { IT_Dictionary1_onLoad(); IT_Dictionary2_onLoad();}
```

- **Create service-independent code**. Testing the code in one service suffices for all services in which that code is used. The code in the previous example is not service-independent. It would fail if the Common_Service_onLoad() function were executed in a service that was missing one or both of the dictionaries. However, this can easily be modified:

```
Common_Service_onLoad () { if (serviceForm.ITDictionary1 !- undefined) {
IT_Dictionary1_onLoad();  }  if (serviceForm.ITDictionary2 !- undefined) {
IT_Dictionary2_onLoad();}
```

Just as the above code tests for the presence of a dictionary in a service before attempting to apply dictionary-specific code, you may need to test for the presence of a particular field before attempting to apply field-specific code. It is best practice to use the dictionary in only one Active Form Component; however, service-specific rules may affect the dictionary's appearance. For example, displaying the supervisor information for the person requesting a service may only be required for those services that require supervisor approval.

Therefore, code that attempts to manipulate the supervisor-related fields should be included in a code block such as:

```
FirstApprover_onLoad () {  if (serviceForm.FirstApprover.SupervisorName !- undefined) {//
   code goes here;  }}
```

A field may be used in a form, but conditionally hidden by a rule or ISF code previously executed. In cases like these, code like the following might be more appropriate, and more robust:

if (serviceForm.FirstApprover.SupervisorName !- undefined) { if (serviceForm.FirstApprover.SupervisorName.isVisible()) {// code goes here; }

- **Send less data to the browser session:** To improve the performance of the form when it is loaded, less data should be sent to the browser session. In previous releases of Service Catalog, any dictionary fields you needed to manipulate had to be sent to the browser in order to be manipulated. Using server-side events, you can manipulate data in dictionaries such that less data is sent to the browser and also it keeps sensitive data under the control of server-side execution and therefore incapable of being intercepted. See Server-Side Data Retrieval Rule, on page 95. For sending less data, you can do the following:
  - Group fields such that fields that do not need to be loaded into the browser session—at least not during the Ordering moment—should be grouped together into one or more "nonloaded" dictionaries. A common example of a "nonloaded" dictionary is one supplying the parameters to an orchestration Agent. The orchestrator often needs more data than the user ever sees on the form. This data is typically derived through conditional rules or data retrieval rules that determine who the user is, what role he has, what OU membership he has, and so on; and all of this is best derived either before the form is loaded in the browser (during the pre-Load event) or once the user has clicked the Submit button (during the post-Submit event).

Seeing the effects of rules operating on "nonloaded" dictionaries will take a bit more testing, since you won't be able to see the values being set until the post-Submit event has occurred. Use of "nonloaded" dictionaries will therefore typically involve checking the form in a subsequent moment (for example, Service Group Authorization or Service Delivery), as a user who has the Manage Service Dictionaries permission and is therefore able to see all the dictionaries defined on the form.

- **Use the pre-Load event rather than the on-Load event**. Generally, conditional rules and data retrieval rules take action on objects that are present in the browser. So a "Hide fields" conditional rule that is tied to the pre-Load event, has in fact nothing to hide, because the fields being hidden do not actually exist yet during the pre-Load event. There is a notable exception to this general rule, as explained in the previous discussion of "nonloaded" dictionaries.

It is possible to manipulate the values of fields that are not loaded into the browser, through either the Set Value conditional rule action, or a data retrieval rule.

The conditional rule and data retrieval rule functionality is highly flexible, enabling you to combine multiple rule actions on multiple targets, and to tie any one rule to multiple events. Any actions that do not have access to their targets—for example, a Set Value action on a field that is not actually present on the service form—are said to "fail silently" or take no effect. If you have a conditional rule that performs a sequence of actions (say, hiding some fields, making others mandatory, and popping up an Alert message; as well as executing a Set Value), you may see most of those actions taking effect in the browser. The effect of a Set Value action, however, may not be visible because either the target field is currently hidden or it is not present in the browser. The conditional rule framework does not stop you from constructing such a rule; it merely ignores any actions that cannot be executed.

The conditional rule and data retrieval rule wizards spell out these behaviors as you construct the rules. Help text embedded into each step explains any limitations you may experience when using certain actions on the server-side events.

- Server-side events provide you with an opportunity to write data directly to the database—specifically, to the WDDX data stored in the database for each requisition entry. The post-Submit event in particular is your window of opportunity for writing the data that was collected or derived from actions in the browser into the database.

An important distinction between the pre-Load and post-Submit events is that pre-Load can write to the database only if the requisition data is already in the database. In other words, if the user has not yet clicked the Order or Add & Review buttons, the data for the requisition does not exist in the database and the pre-Load rules have nothing to manipulate. While the pre-Load event gives you the ability to change the viewable value, it does not persist that value in the database. The post-Submit event provides the persistence.

- Server-Side events provides you with great flexibility for manipulating data that is part of the service form yet protected by the server. What makes this possible is that the "nonloaded" dictionaries are stored in the database and therefore accessible by server-side rules. Although the browser-side rules are generated as JavaScript on the form, the server-side equivalents of those rules are actually generated as Java code—so there are, in effect, two separate manifestations of rules possible. The ISF framework (documented in the Interactive Service Forms (ISF) API Overview, on page 61) enables you to write JavaScript for more complex manipulations of the form data and appearance. This JavaScript can only execute in the browser. Therefore any ISF functions you write cannot be tied to the server-side events.
- This **Editable on server-side only** option, available on the HTML Representation tab for any field defined as Input Type = read-only or hidden, helps you satisfy what are sometimes conflicting requirements: the first is to display helpful data to the requesting user or use "hidden variables" on the form that drive business logic; and the second is to secure data completely from user intervention (including malicious activity).

Two common use-cases for using this flag are:

- ◦ The person-based dictionary, a "template dictionary" you can create in the Dictionaries module
  ◦ The service-item-based dictionary, the structure of which is also automatically created in the Dictionaries module

The two are similar in that they can both exhibit "autopopulation" behavior. In the person-based dictionary, the form user chooses a person (using the Select Person control) and that person's details are automatically displayed on the form. In the service-item-based dictionary, the form user can choose a particular service item he owns or has access to, and the attribute values for that service item instance are automatically displayed.

It is a common service design technique to define most (if not all) of the attributes automatically populated as read-only fields (that is, configured with an Input Type on the HTML representation tab as "read-only"). For example, if your user base is large enough to contain multiple people with the same name, you may use attributes such as the email address to help the form user differentiate among those people and choose the right person. Of course you would not expect the form user to be updating the chosen user's email address. On the other hand, the form user may know this person well and therefore know that he uses his mobile phone exclusively and not the desk phone number obtained through Directory Integration. If this is the case, you may want to make the Work Phone field editable and not just read-only, so that at least the service delivery personnel handling the request have the most reliable way of reaching the chosen person.

Using the "Editable on server-side only" check box ensures that any fields you mark as Input Type = read-only or hidden are entirely controlled by the server and that any manipulation of their values in the browser session

is discarded in favor of the data residing in the Service Catalog database. In fact the only mechanism to override the data residing in the database is a rule (either conditional or data retrieval) that executes during a server-side event, and in this case that rule is actually updating the database value. (In the case of the person-based dictionary, the Person record in DirPerson is not updated, but the WDDX data stored in the database for the requisition entry is.)

In other words, you—the service designer—can manipulate the value of a field marked as "Editable on server-side only," but you can do so only through rules executing during server-side events. This is an important factor to consider, because you may be tempted to think that all you have to do to use these "protected" fields is to check the "Editable on server-side only" flag. That is all that's required if you do not have any reason to manipulate the value returned from the database. But if your business logic requires that you set these values in response to selections the form user makes, you must be sure to tie any conditional rule (for a Set Value action) or data retrieval rule (for a data distribution from a query) to a server-side event.

- **Naming Conventions**: Although it is perfectly legitimate, you should not give a dictionary, form, and field the same name. It results in a very confusing display. Dictionaries should ideally use a prefix notation to denote their usage and perhaps the dictionary group in which they have been placed. For example, a "Reason" dictionary sounds like it is fairly generic, used in many forms. Therefore, it makes sense to place it in a dictionary group named, "Common", and to prefix the dictionary name with a code designating this dictionary group, for example, CMN_Reason. Dictionary groups could reflect the service group in which a service-specific dictionary is used, or perhaps the company department or division for which the dictionary and service have been developed.

An easily understood naming convention for forms is also needed. This is especially critical if you need to differentiate between multiple forms that include the same dictionary or group of dictionaries.

- **Form-Dictionary Relationship**: One form should have only one dictionary. Multiple dictionaries should be included in the same form if:
  - These dictionaries will all be required in the same services.
  - The order in which the dictionaries are presented is the same in all services.
  - No additional dictionaries need to be displayed between the dictionaries in this form. On a service form, all dictionaries in one form component are displayed, in the order specified in that component; then, dictionaries in the next form component included in the service are displayed, in the order specified, and so on.

Examples:

- Assume that a group of services developed for the Facilities department have a chain of approvals consisting of up to three approvers, based on the customer's position or department. In this case the Facilities_Approval form should include three dictionaries: FirstApprover, SecondApprover, and ThirdApprover, as well as rules to determine how many of the three approvals actually need to be collected.
- Assume that most of the services developed for the Facilities department require the standard up-to-three approvers, but a rather expensive service requires an additional approver, at the VP level. You have two options:

One Form	Modify the Facilities_Approval form so that it includes the VP Approver dictionary in addition to the FirstApprover, SecondApprover, and ThirdApprover, and write an additional rule or rules to display and process the VPApprover dictionary only for that one service.
Multiple Forms	Create another form, Facilities_VP_Approval, which includes the VPApprover dictionary and duplicates the HTML, access control, and rules needed to process the FirstApprover, SecondApprover, and ThirdApprover dictionaries.

In most cases, the first solution (one slightly more complex form) should be preferred. You don't have to duplicate work to configure three dictionaries and their associated rules in more than one form. And, if any of those dictionaries or their rules needs to be changed, there is only one place to change them. You should only consider the second solution (two partly redundant forms) when including the additional dictionary would also cause extensive changes to the way the other dictionaries are displayed or processed.

- **Different Renderings for the Same Dictionary**: What if the form's behavior or appearance needs to vary greatly, based on the services in which the form is used? Various scenarios are available. Only you, the service designer, can decide which scenario is preferable, based on the criteria outlined below. For example, virtually every service needs to include that “Reason” dictionary. But sometimes the field label should read “Reason”, sometimes “Justification”, sometimes “Explanation”. Further, the help text associated with the field needs to be substantially different for each service or group of services. In this case, since the dictionary is simple and easily configured, a decision is easy: Create a separate form for each rendering of the dictionary, and include the appropriate form in the corresponding services.

But what if the field or fields that need to be customized on a service-by-service basis were part of a large dictionary with potentially complex rules? You still have the same two options outlined above: one form or multiple forms. However:

One Form	Create one form and customize its appearance using ISF functions.
Multiple Forms	Create a different form for each rendering of the dictionary.

The One-Form option is now complicated by the fact that you cannot customize the dictionary's appearance using conditional rules—conditional rules only modify a field's appearance and behavior, not the contents of the field's label and help text. The Multiple-Forms option has the same drawbacks as before—you need to do the work upfront to create the forms, and the maintenance work to maintain rules in multiple places.

There are two additional options:

- ◦ Get the requirements changed. This is not likely nor a good idea, if it results in the users getting less information about how to supply information in requesting a service.

- Put the problematic field in a separate dictionary that is used in multiple forms, and keep one form for the rest of the dictionary. This is only possible if the field can be displayed before or after all the other fields, not in the middle.

- **Dictionaries, Forms, Rules and Services:** A form typically consists of a set of dictionaries and a set of rules.
  - The rules may affect not only those dictionaries (and their fields) in the same form but dictionaries in other forms.
  - It is very important to test the rules thoroughly in the context of a particular service, to ensure that all referenced dictionaries are in forms included in that service. The Best Practices Reports include a "Dictionaries in Services" report that will facilitate doing this testing and doing an impact analysis to assess potential effects of any proposed changes to dictionaries or rules.

It is also perfectly reasonable to structure Form 2 so that it contains only rules to be applied to a particular service. Many services could include only Form 1, but only the service with extra requirements includes Form 2. This greatly simplified (or potentially eliminates) conditions that would be needed in the rules in Form 2, to ensure they only fire in the appropriate services.

This also eliminates having to write a conditional rule that tests for the service name—a textual element that is subject to change. You would merely have to include the rules-only form in the service affected, sequencing it after the form whose dictionaries are affected. Rules are executed in the order in which the forms are included in the service, and then in the order in which the rules are arranged within the form.

- **Loosely Coupled vs. Tightly Coupled Rules and Dictionaries:** When rules and the dictionaries they affect and refer to are in the same AFC, the rules and dictionaries are said to be "tightly coupled". The example above, with a rule affecting the appearance of a dictionary in another form, shows "loosely coupled" rules and dictionaries. In general, a rule can affect a dictionary in any form, provided both forms are included in the same service. There is, however, one significant limitation on loosely coupling rules and dictionaries—a rule defined in one form cannot be triggered by an field-level event attached to a field in a dictionary in another form. Therefore, loosely coupled rules typically need to be triggered by a form-level event, when the form is loaded or submitted.
- **Rules, ISF and the Requisition Life Cycle**: When a request is submitted, all of the definitions for the dictionaries, HTML representation, access control, and task plan for that service are stored (in a compressed format) with the request data. This ensures that a request can be processed throughout the authorization and delivery moments without any subsequent changes to the form's definition affecting the behavior or appearance of the service form for the in-flight request.

If you change a dictionary, all forms using that dictionary automatically inherit the change and all services which use those forms also inherit the change. Any request for one of the changed services that has been saved but not submitted are marked as obsolete. The user will need to cancel the request or remove the service, read it, and fill in the service form data. Submitted requests are not affected by changes to dictionaries or active form components.

However, rules and ISF functions (whether in a Script or an external library) are always loaded dynamically when a service form is displayed in My Services or Service Manager. Therefore, you must ensure that a current version of the rule/code does not refer to a field no longer on the form or, conversely, that a field on previous versions of the form, but not longer used, can have an associated function. This flexibility is easily accomplished by always checking for the existence of a dictionary or field before attempting to manipulate it via ISF, as discussed previously. If desired, a different version of a library could be attached to the newer versions of the

form so that, for example, the names of functions would not need to change, but their contents could be updated to comply with revised requirements.

- **Changing a Dictionary or Active Form Component**: If you change the definition of a field within a dictionary, all active form components will reflect that change. Similarly, if you change any aspect of the active form component's definition, all service definitions that use that form will reflect that change. You cannot delete/change a dictionary or dictionary field to which a conditional rule is associated, or that triggers a data retrieval rule; you must remove the associations first. This check is not performed for JavaScript functions. Each time the form is changed, the application automatically updates the version number for all service definitions which incorporate that form. You may see a brief delay in saving such changes if the form is used within many services. This may also affect any Requisition API (RAPI) programs implemented for ordering the service, since the RAPI SubmitRequest operations requires the version number of the service to be specified.
- **Coding SQL Entry Data Retrieval Rules:** If you are not familiar with SQL, it may be a bit intimidating to code a SQL Entry data retrieval rule. A "trick" to help familiarize yourself with SQL is to start by configuring a simpler version of the rule as a Database Table Lookup. After you have saved the rule, the rule summary displays the generated SQL. You can copy the SQL statement, paste it into a SQL Entry data retrieval rule, and modify it as required.
- **Using Customer and Initiator Lightweight Namespaces:** The #Customer# and #Initiator# lightweight namespaces are automatically supplied as the default values for fields in the Customer_Information and Initiator_Information dictionaries used in the Customer-Initiator form component. However, these namespaces could be used as default values for any form component.

The design of dictionaries to hold information about customer location could use this technique. Fields about the customer's location could be included in the Customer_Information dictionary. However, such fields may be required on very few services—only those where a service must be delivered to the customer's physical location. A more flexible design may be to configure a separate dictionary (and form component) for the customer location, and include this form component only in services where customer location is relevant.

Multiple namespaces can be specified as a default value. For example, the expression #Customer.FirstName# #Customer.LastName# concatenates the customer's first name and last name, separated by one space. This expression duplicates the appearance of the first field of a person-based dictionary.

Note The use of grid dictionary fields for lightweight namespaces is not currently supported.

Server-Side Data Retrieval Rule

Data retrieval rule's triggering event can be server-side or browser-side. There are two form-level events that execute server-side—pre-Load and post-Submit. You can use these events to execute rules on the server. Such rules are commonly referred to as "server-side rules". As opposed to client-side rules that act upon the service form loaded on the browser, server-side rules act upon any dictionaries in the service form. This includes dictionaries that are never sent to the browser (that is, to which users have neither View nor Edit permissions).

The powerful behavior described above means that if you attach one or more data retrieval rules to the pre-Load event, those rules take effect on all dictionaries, for all system moments. Therefore, if you create a form rule intended to prefill data at the Ordering moment, but be ignored in subsequent server-side events when the dictionaries become read-only, you should not associate it with server-side events. The form is rendered in the browser only after the pre-Load data retrieval completes.

This often provides a better user experience as the time between initial rendering of the service form on the browser and the completion of client-side events can be reduced. However, depending on the size and complexity of the form, the end user's perception of the form load may be better if you minimize the pre-Load data retrieval.

Unlike client-side rules triggered at form load time, form rules that are triggered at pre-Load time are not re-executed when the form is refreshed after hitting validation errors.

Importing Service Items and Standards using Service Link

You can use a Service Link agent to import either service items or standards. This allows you to include the import step as a task in a standard service request. The agent will import a file formatted as described in Import File Format For Service Items and Standards, on page 97, and supports the same import options as are available via the Import from File utility. The message is logged in Service Link as a "SIM Import" message type.

For detailed instructions on using Service Link, see the Cisco Prime Service Catalog Adapter Integration Guide .

To configure a Service Link agent to import service items or standards:

1. Define a new agent in Service Link, specifying a "Service Item" task as the Context Type. At a minimum, you will need separate agents for standards and service items.
2. Specify the "Dummy" adapter as the outbound adapter, and the "File" adapter as the inbound adapter. No transformations are required.
3. Skip the pages on configuring the outbound adapter properties and its parameters.
4. For the inbound adapter properties, specify the names of the directories to be used for processing the import file. The file should be written to the designated "Input" directory.
5. For the inbound parameters, create four parameters and enter the appropriate value in the Dictionary Field as shown in the table below. (Parameter values are case sensitive.)

Table 24: Inbound Parameters and Dictionary Values Table

Parameter Name	Valid Values	Description
Domain	SERVICE ITEM STANDARD	An import file can contain data or definition of service item.
ImportDefinition	TRUE FALSE	True if the definition portion (if any) of the import file should be processed; false otherwise.
ImportData	TRUE FALSE	True if the data portion (if any) of the import file should be processed; false otherwise.
ConflictResolution	INSERT OVERWRITE MERGE	Behavior of the import process in reconciling new data or definitions, in the input file, with data or definitions currently in Service Catalog.

Import File Format For Service Items and Standards

Each import file is an XML file in an industry-standard CIM (Common Information Model) compatible format, version 2.3.1. CIM is based on an object-oriented model and uses technology adapted from Unified Modeling Language (UML). A Document Type Definition (DTD) describing the file format used by Service Catalog can be found at RequestCenter.war\DTD on the application server.

Service Catalog recognizes only a subset of entities that are available under CIM. Those entities that it does not recognize it ignores. In Service Catalog's CIM implementation:

- Each service item or standard is a **class**.
- The details about the service item—its description, display name, and assigned group (classification)—are **qualifiers** for the service item.
- The attributes specified for the service item are **properties** of the service item class.
- Each attribute (property), in turn, has **qualifiers**. These qualifiers constitute the detailed configuration for each attribute—its description, caption, whether it is visible in My Services, and the sequence in which the attribute occurs in the service item definition.
- Each service item class may have many **instances**. Each of these corresponds to one service item instance.
- Each instance of a service item has one **property** for each attribute of the service item. You may also include, optionally, three attributes that comprise the subscription information for the service item during import—Requisition Entry ID, Customer Login Name, and Organizational Unit Name. The value in the XML file updates the value of the corresponding attribute or subscription field of the service item instance.

The previous discussion applies to both service items and standards in general. Note that standards are not "owned" by any individual user or organizational unit per se and therefore you cannot set subscription attributes for them.

Syntax for a Service Item Import File

The Table 25: Syntax for Specifying a Service Item or Standards Import File table summarizes the syntax for specifying a service item or standards import file. The same syntax applies to both service items and standards. Each XML file consists of one SI Import Specification.

The Import Specification (SIImportSpec), in turn, consists of a SIClassDefinition, followed by a list of one more SIInstances for the specified class (Service Item or standard). One or both of the class definition and instances may be included in the file. File content should match the import options specified.

Each SIClass specification consists of the service item/standard definition, embedded within appropriate XML tags, followed by the definition of each of the attributes.

Each attribute definition specifies the name, caption, and other configuration options specified for the attribute.

Each service item instance (SIInstance) specifies the name of the service item/standard to which the instance applies and a list of the values for each of the attributes of the item. If an attribute value is blank, appropriate tags must still be included in the import file, with no value specified for the attribute.

Table 25: Syntax for Specifying a Service Item or Standards Import File

SIImportSpec =	S<?xml version="1.0" encoding="utf-8"?> <CIM CIMVERSION="" DTDVERSION=""> <DECLARATION> <DECLGROUP> SIClassDefinition SIInstanceList </DECLGROUP> </DECLARATION> </CIM>
SIClassDefinition =	<<VALUE.OBJECT> SIClass </VALUE.OBJECT>
SIClass =	<CLASS NAME="Service Item Name"> <QUALIFIER NAME="Description" TYPE="string"> <VALUE></VALUE> </QUALIFIER> <QUALIFIER NAME="Classification" TYPE="string"> <VALUE>Service Item Group</VALUE> </QUALIFIER> <QUALIFIER NAME="DisplayName" TYPE="string"> <VALUE>SI Display Name</VALUE> </QUALIFIER> SIAttributeList </CLASS>
SIAttributeList =	SIAttributeDefinition [SIAttributeDefinition]

SIAttributeDefinition =	<PROPERTY NAME="Name" TYPE="string"> <QUALIFIER NAME="Description" TYPE="string"> <VALUE>Description of the attribute</VALUE> </QUALIFIER> <QUALIFIER NAME="Caption" TYPE="string"> <VALUE>Caption for the attribute</VALUE> </QUALIFIER> <QUALIFIER NAME="IsVisible" TYPE="sint32"> <VALUE>1 if visible, 0 if not visible</VALUE> </QUALIFIER> <QUALIFIER NAME="DisplayOrder" TYPE="sint32"> <VALUE>Sequence of the attribute</VALUE> </QUALIFIER> </PROPERTY>
SIInstanceList =	SIInstance [SIInstance]
SIInstance =	<VALUE.OBJECT> <INSTANCE CLASSNAME="ClassName"> SIPropertyList </INSTANCE> </VALUE.OBJECT>
SIInstanceDataList=	SIInstanceData [SIInstanceData]

SIInstanceData=	<PROPERTY NAME="AttributeName" TYPE=SIDataType> <VALUE>Attribute value</VALUE> </PROPERTY> Optional subscription properties: <PROPERTY NAME="#RequisitionEntryID#" TYPE="sint32"> <VALUE>Requisition entry ID</VALUE> </PROPERTY> <PROPERTY NAME="#LoginName#" TYPE="string"> <VALUE>Customer login name</VALUE> </PROPERTY> <PROPERTY NAME="#OrganizationalUnitName#" TYPE="string"> <VALUE>Organizational unit name</VALUE> </PROPERTY> <PROPERTY NAME="#AccountName#" TYPE="string"> <VALUE>account name</VALUE> </PROPERTY> <PROPERTY NAME="#Operation#" TYPE="string"> <VALUE></VALUE> </PROPERTY>
SIDataType =	{“char16” | “string” | “datetime” | “real64” | “sint32” | “sint64”}

The datatypes do not exactly match the datatypes specified when you define a Service Item or Standard using Service Item Manager. Map the Service Item Manager datatypes to those used in the import file using the Table 26: Import File Datatype for Service Item Manager Datatype table.

Table 26: Import File Datatype for Service Item Manager Datatype

Service Item Manager Datatype	Import File Datatype
STRING(32)	char16
STRING(128)	<not supported>
STRING(512)	string

Service Item Manager Datatype	Import File Datatype
INTEGER	sint32
LONGINTEGER	sint64
DOUBLEFLOAT	real64
MONEY	Use real64
DATETIME	Datetime (must use the following format for date value = yyyy-mm-dd hh:mm:ss, for example 2009-04-15 12:00:00)

Service Item Subscription Processing Rules for Imported File

Unlike create and update operations performed by internal and external service item tasks, the create and update operations handled through the file import mechanism happen outside the context of a service request delivery plan. Hence the import program does not validate the relationship among the Requisition Entry ID, Customer Login Name, and Organizational Unit Name provided in the import file. As long as they are valid IDs/names, the input is accepted.

Here are the high-level processing rules for subscription:

- If no subscription information is provided when a service item instance is created, the item stays unassigned.
- If only Customer Login Name property is specified at the time a service item instance is created, the Organizational Unit owner of the item is set to the home organizational unit of the customer and the account ownership is set to the tenant account derived from the home organizational unit if there is one.
- If Customer Login Name or Organizational Unit name, or Account name properties are specified, the values are used to update the service item subscription. When the value is blank, the corresponding subscription field is set to null. The absence of a property in the import file means no change/override on the property value.
- When a Requisition Entry ID is provided in the properties, the corresponding requisition ID is associated with the service item and displayed in My Services and Service Item Manager.
- Once a Requisition Entry ID is set for a service item, it cannot be modified or reset to null.

The possible combinations for customer and organizational unit assignment and the outcome are summarized in Table 27: Service Item Subscription Table.

Account assignment works in a similar way as organizational unit assignment and the permutations are not enumerated in the table below.

:

Table 27: Service Item Subscription Table

When service item is created			
Property In XML	Resulting Subscription		
Login Name	OU Name	Customer	OU
None	None	NULL	NULL
None	Blank or Invalid Value	NULL	NULL
None	Valid OU Name	NULL	OU provided
Blank or Invalid Value	None	NULL	NULL
Blank or Invalid Value	Blank or Invalid Value	NULL	NULL
Blank or Invalid Value	Valid OU Name	NULL	OU provided
Valid Customer	None	Customer provided	Home OU of Customer
Valid Customer	No Value	Customer provided	NULL
Valid Customer	Valid OU Name	Customer provided	OU provided

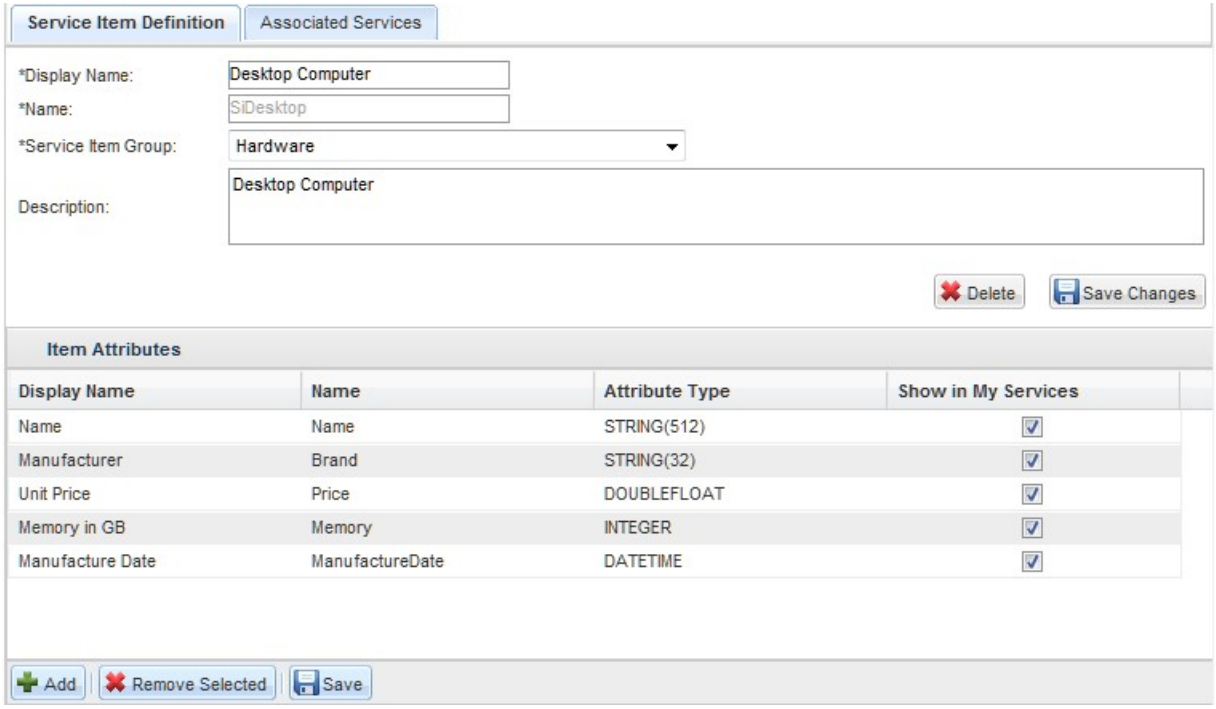
When service item is updated			
Property In XML	Resulting Subscription		
Login Name	OU Name	Customer	OU
None	None	No change	No change
None	Blank or Invalid Value	No change	NULL
None	Valid OU Name	No change	OU provided
Blank or Invalid Value	None	NULL	No change
Blank or Invalid Value	Blank or Invalid Value	NULL	NULL
Blank or Invalid Value	Valid OU Name	NULL	OU provided
Valid Customer	None	Customer provided	No change
Valid Customer	Blank or Invalid Value	Customer provided	NULL

When service item is updated			
Valid Customer	Valid OU Name	Customer provided	OU provided

Example: Service Item Import File

Assume that a “Desktop” service item has been created, with the following definition:

Figure 11: Desktop Service Item



The import file for this service item might look like the sample below:

Table 28: Import File for Service Item Example

Sample XML	Description/Usage
<?xml version=“1.0” encoding=“utf-8”?> <CIM CIMVERSION=“” DTDVERSION=“”> <DECLARATION> <DECLGROUP>	Beginning of SI Import Specification.

Sample XML	Description/Usage
< VALUE.OBJECT> <CLASS NAME="Desktop"> <QUALIFIER NAME="Description" TYPE="string"> <VALUE>Desktop Computer.</VALUE> </QUALIFIER> <QUALIFIER NAME="Classification" TYPE="string"> <VALUE>Hardware</VALUE> </QUALIFIER> <QUALIFIER NAME="DisplayName" TYPE="string"> <VALUE>Desktop Computer</VALUE> </QUALIFIER>	Properties of an SI—its name, description, display name, and group (classification).
<PROPERTY NAME="Name" TYPE="string"> <QUALIFIER NAME="Description" TYPE="string"> <VALUE>The name of the computer.</VALUE> </QUALIFIER> <QUALIFIER NAME="Caption" TYPE="string"> <VALUE>Name</VALUE> </QUALIFIER> <QUALIFIER NAME="IsVisible" TYPE="sint32"> <VALUE>1</VALUE> </QUALIFIER> <QUALIFIER NAME="DisplayOrder" TYPE="sint32"> <VALUE>1</VALUE> </QUALIFIER> </PROPERTY>	Definition of the Name attribute of the SI. The Name attribute is required in all SIs; its caption must also be "Name".

Sample XML	Description/Usage
<PROPERTY NAME=“Brand” TYPE=“char16”> <QUALIFIER NAME=“Description” TYPE=“string”> <VALUE>Brand name of the computer.</VALUE> </QUALIFIER> <QUALIFIER NAME=“Caption” TYPE=“string”> <VALUE>Manufacturer</VALUE> </QUALIFIER> <QUALIFIER NAME=“IsVisible” TYPE=“sint32”> <VALUE>1</VALUE> </QUALIFIER> <QUALIFIER NAME=“DisplayOrder” TYPE=“sint32”> <VALUE>2</VALUE> </QUALIFIER> </PROPERTY>	Definition of the Brand attribute of the SI.
<PROPERTY NAME=“Price” TYPE=“real64”> <QUALIFIER NAME=“Description” TYPE=“string”> <VALUE>MSRP</VALUE> </QUALIFIER> <QUALIFIER NAME=“Caption” TYPE=“string”> <VALUE>Unit Price</VALUE> </QUALIFIER> <QUALIFIER NAME=“IsVisible” TYPE=“sint32”> <VALUE>1</VALUE> </QUALIFIER> <QUALIFIER NAME=“DisplayOrder” TYPE=“sint32”> <VALUE>3</VALUE> </QUALIFIER> </PROPERTY>	Definition of the Price attribute of the SI.

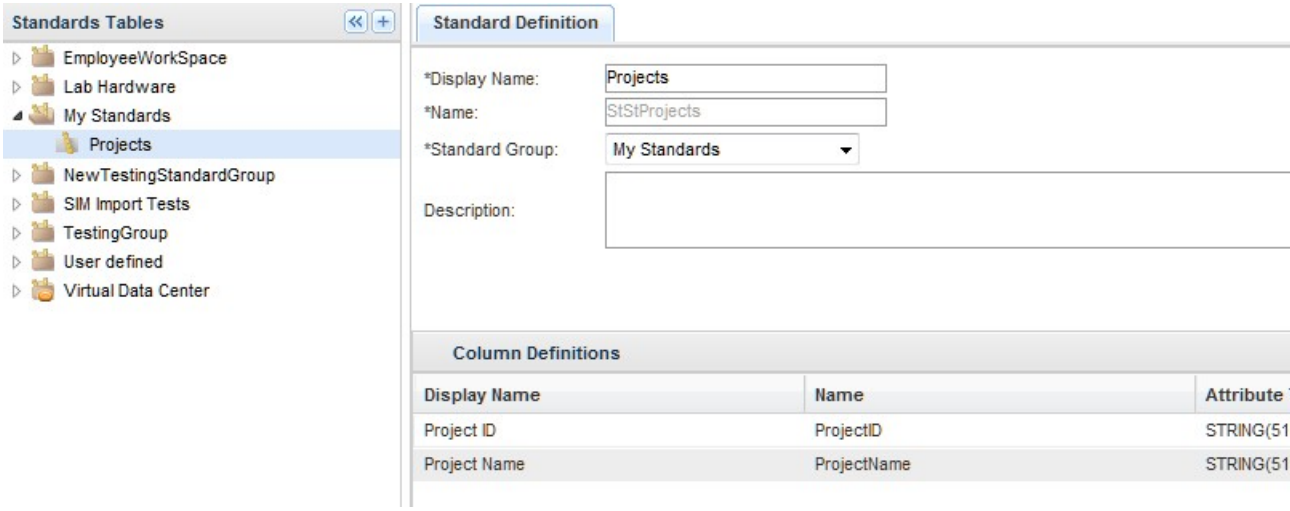
Sample XML	Description/Usage
<PROPERTY NAME="Memory" TYPE="sint32"> <QUALIFIER NAME="Description" TYPE="string"> <VALUE>Amount of RAM in GB</VALUE> </QUALIFIER> <QUALIFIER NAME="Caption" TYPE="string"> <VALUE>Memory in GB</VALUE> </QUALIFIER> <QUALIFIER NAME="IsVisible" TYPE="sint32"> <VALUE>1</VALUE> </QUALIFIER> <QUALIFIER NAME="DisplayOrder" TYPE="sint32"> <VALUE>4</VALUE> </QUALIFIER> </PROPERTY>	Definition of the Memory attribute of the SI.
< PROPERTY NAME="ManufactureDate" TYPE="datetime"> <QUALIFIER NAME="Description" TYPE="string"> <VALUE>Date of manufacture</VALUE> </QUALIFIER> <QUALIFIER NAME="Caption" TYPE="string"> <VALUE>Manufacture Date</VALUE> </QUALIFIER> <QUALIFIER NAME="IsVisible" TYPE="sint32"> <VALUE>1</VALUE> </QUALIFIER> <QUALIFIER NAME="DisplayOrder" TYPE="sint32"> <VALUE>5</VALUE> </QUALIFIER> </PROPERTY> </CLASS> </VALUE.OBJECT>	Definition of the ManufactureDate attribute of the SI.

Sample XML	Description/Usage
<VALUE.OBJECT> <INSTANCE CLASSNAME=“Desktop”> <PROPERTY NAME=“Name” TYPE=“string”> <VALUE>Thinkpad T60</VALUE> </PROPERTY> <PROPERTY NAME=“Brand” TYPE=“char16”> <VALUE>LENOVO</VALUE> </PROPERTY> <PROPERTY NAME=“Price” TYPE=“real64”> <VALUE>899.99</VALUE> </PROPERTY> <PROPERTY NAME=“Memory” TYPE=“sint32”> <VALUE>3</VALUE> </PROPERTY> <PROPERTY NAME=“ManufactureDate” TYPE=“datetime”> <VALUE>2009-04-15 12:00:00</VALUE> </PROPERTY> <PROPERTY NAME="#RequisitionEntryID#" TYPE="sint32"> <VALUE>10</VALUE> </PROPERTY> <PROPERTY NAME="#LoginName#" TYPE="string"> <VALUE>jsmith</VALUE> </PROPERTY> <PROPERTY NAME="#OrganizationalUnitName#" TYPE="string"> <VALUE>Finance</VALUE> </PROPERTY> </INSTANCE> </VALUE.OBJECT>	One instance of a Desktop SI, for a “Thinkpad T60”.
</DECLGROUP> </DECLARATION> </CIM>	End of the SI Import Specification.

Example: Standard Import File Example

Assume that a “Projects” standard has been created, with the following definition:

Figure 12: Standard Import File Example



The Service Link Service Item Task would be defined with the following agent parameters:General

- Outbound Properties
- Inbound Properties
- Outbound Request Parameter
- Inbound Response Parameter
- Internal Parameters

The import file for this standard might look like the sample below:

Table 29: Standard Import File Example

Sample XML	Description/Usage
<?xml version=“1.0” encoding=“utf-8”?> <CIM CIMVERSION=“” DTDVERSION=“”> <DECLARATION> <DECLGROUP>	Beginning of Standard Import Specification.

Sample XML	Description/Usage
< VALUE.OBJECT> <CLASS NAME="Projects"> <QUALIFIER NAME="Description" TYPE="string"> <VALUE>Describe it here</VALUE> </QUALIFIER> <QUALIFIER NAME="Classification" TYPE="string"> <VALUE>My Standards</VALUE> </QUALIFIER> <QUALIFIER NAME="DisplayName" TYPE="string"> <VALUE>Projects</VALUE> </QUALIFIER>	Properties of the Standard—its name, description, display name, and group (classification).
<PROPERTY NAME="ProjectID" TYPE="string"> <QUALIFIER NAME="Description" TYPE="string"> <VALUE>ID of the Project</VALUE> </QUALIFIER> <QUALIFIER NAME="Caption" TYPE="string"> <VALUE>Project ID</VALUE> </QUALIFIER> <QUALIFIER NAME="IsVisible" TYPE="sint32"> <VALUE>1</VALUE> </QUALIFIER> <QUALIFIER NAME="DisplayOrder" TYPE="sint32"> <VALUE>1</VALUE> </QUALIFIER> </PROPERTY>	Definition of the ProjectID attribute of the Standard.

Sample XML	Description/Usage
<PROPERTY NAME="ProjectName" TYPE="string"> <QUALIFIER NAME="Description" TYPE="string"> <VALUE>Name of the project</VALUE> </QUALIFIER> <QUALIFIER NAME="Caption" TYPE="string"> <VALUE>Project Name</VALUE> </QUALIFIER> <QUALIFIER NAME="IsVisible" TYPE="sint32"> <VALUE>1</VALUE> </QUALIFIER> <QUALIFIER NAME="DisplayOrder" TYPE="sint32"> <VALUE>2</VALUE> </QUALIFIER> </PROPERTY>	Definition of the ProjectName attribute of the Standard.
<VALUE.OBJECT> <INSTANCE CLASSNAME="Projects"> <PROPERTY NAME="ProjectID" TYPE="string"> <VALUE>001</VALUE> </PROPERTY> <PROPERTY NAME="ProjectName" TYPE="string"> <VALUE>Reporting System Upgrade</VALUE> </PROPERTY> </INSTANCE> </VALUE.OBJECT>	First instance of a Project Standard.

Sample XML	Description/Usage
< VALUE.OBJECT> <INSTANCE CLASSNAME="Projects"> <PROPERTY NAME="ProjectID" TYPE="string"> <VALUE>002</VALUE> </PROPERTY> <PROPERTY NAME="ProjectName" TYPE="string"> <VALUE>Decision Support System Installation</VALUE> </PROPERTY> </INSTANCE> </VALUE.OBJECT>	Second instance of a Project Standard.
</DECLGROUP> </DECLARATION> </CIM>	End of the Standards Import Specification.

Service Item Import DTD

The Document Type Definition (DTD) that describes the format required for a Service Item or Standards import file is available on the Service Catalog application server, under RequestCenter.war\DTD. The DTD is given below.

```
<!ENTITY % CIMName "NAME           CDATA         #REQUIRED">
<!ENTITY % CIMType "TYPE (char16|string|sint32|sint64|datetime|real64)">
<!ENTITY % ClassName "CLASSNAME      CDATA         #REQUIRED">
<!ELEMENTCIM (DECLARATION)>
<!ATTLISTCIM
 CIMVERSION CDATA #REQUIRED
 DTDVERSION CDATA #REQUIRED
>
<!ELEMENTDECLARATION (DECLGROUP)+>
<!ELEMENTDECLGROUP (VALUE.OBJECT*)>
<!ELEMENTVALUE.OBJECT (CLASS | INSTANCE)>
<!ELEMENTCLASS (QUALIFIER*, PROPERTY*, METHOD*)>
<!ATTLISTCLASS
 %CIMName;
>
<!ELEMENTQUALIFIER (VALUE?)>
<!ATTLISTQUALIFIER %CIMName;
         %CIMType;              #REQUIRED
>
<!ELEMENTPROPERTY (QUALIFIER*, VALUE?)>
<!ATTLISTPROPERTY %CIMName;
         %CIMType;              #REQUIRED
>
<!ELEMENTVALUE (#PCDATA)>
<!ELEMENTINSTANCE (QUALIFIER*, PROPERTY*)>
<!ATTLISTINSTANCE
 %ClassName;
```

```
>
<!ELEMENTMETHOD (QUALIFIER*)>
<!ATTLISTMETHOD %CIMName;
>
```