servicenow

Professional Developer Guide to the Now Platform

Guide to developing custom business applications on the Now Platform

servicenow

Table of Contents

Introduction

ServiceNow provides a single mobile and web application development platform to quickly build business applications to power digital transformation. Learn how anyone can automate, extend, and build digital workflow apps across the enterprise with the Now Platform.

- **Plan**: Think through the application before building it
- **Build**: Create and configure the application and its components
- **Validate**: Perform functional testing and write test cases
- **Deploy**: Move the application into production

The remainder of this guide provides guidance for each of these steps to help developers easily build, test, and deploy a ServiceNow application.

Plan

Paramount in building any app is the planning process. Being *intentional* about the planning step has both immediate and long-term benefits for the app.

Candidates for a good ServiceNow app

Not every application idea is a good fit on the Now Platform. Some apps are better suited for the Now Platform than others:

Good Fit	Poor Fit
• Simple forms • Task management • Request fulfillment • Excel driven processes • Repeatable processes • 3rd party integrations • Orchestration of multiple systems • Single experience from functions in multiple systems • Web and Mobile access to the same apps and data simultaneously	• Unstructured data • Unrepeatable processes • Requires graphics processing • Streaming audio or video • Highly customized UI

Pre-Build Decisions

Some actions taken when building an app are irreversible. Plan for these topics in advance.

- Scoped vs. Global Applications
- Instance Selection
- Naming Decisions

Scoped vs. Global Applications

One of the first major decisions to make when creating an app is: *Should the app be in its own private scope or in the global scope?*

By default, apps are created in their own private application scope. Applications in a private application scope restrict access to their application artifacts so that only application artifacts in the same scope have full access to create, modify, remove, or run application data. Scoped applications can use source control integration and delegated development. Globally scoped applications cannot integrate with source control or use delegated development.

Create custom business applications in scope unless:

- the application has to delete global data.
- the application needs to change application access settings on multiple default tables to function.
- the application needs to access APIs only available in the global scope and creating a globally scoped passthrough script include is not enough for this requirement. A globally scoped passthrough is a script include created in a global scope that is accessible from the private scope and gives access to a global API that is not accessible by default from a private application scope.

DOCUMENTATION: Application Scope

COMMUNITY WHITEPAPER: Understanding Application Scope on the Now Platform

Start in the Appropriate Instance

Proof of Concept (PoC) application builds can and should be built in a separate instance from a regular development instance. The instance can be a sandbox instance or a personal developer instance (PDI) from the Developer Portal [developer.servicenow.com]. PDIs are named in the format of dev12345.service-now.com.

If utilizing an instance with a different scope namespace, rebuild the PoC apps in the organization's development instance, but do not import them into that instance as the scoped namespace on the app will not match that of the company's development instance.

Applications that the organization intends to use (i.e., production apps) should be created in the organization's dev instance so the application can follow the organization's testing and deployment process.

Naming Decisions

The application *Name* matters. ServiceNow suggests a *Scope* based on the application's *Name*. Application file names are appended to the *Scope* to uniquely identify application resources in an instance. Scope has the form of x*_[company code]_[app_name]* with a maximum of 18 characters. For example, an application name **Legal Request** has a suggested scope of **x_acme_legal_reque**.

Because all application files within the app inherit the Scope, carefully consider what the value should be. The Name of the application can always be changed.

Prerequisites for building an app

The following are required to build an app:

- A ServiceNow instance
- An admin or delegated developer role in that ServiceNow instance. A delegated developer role is a role with fewer privileges than the admin role that still allows for application development.

See Appendix B for recommended training when onboarding a new developer.

Managing Development

Source Control

To enable simple, future-proof collaboration for your app, use Source Control Integration. Source control allows an organization to adopt industry standard toolsets while following modern application development and team management paradigms. See the ServiceNow documentation for more on Source Control Integration.

While source control platforms, such as Git and GitHub, align an organization with current and future best-practice trends in ServiceNow application development and management, some organizations might not be ready to adopt the current source control integration:

- Branch merging is not currently supported
- Corporate Git repositories behind a firewall cannot be directly integrated without additional firewall configuration
- Global applications are not currently supported

For more guidance on developing as a team on the Now Platform reference the Developing as a Team Guide on the developer site.

Delegated Development

Delegated Development allows designated users without the *admin* role to develop or deploy applications on the Now Platform. Users with the application-specific admin role or the system-level *admin* role can delegate application development to designated developers at the application level.

If possible, use delegated development and give developers the lowest set of privileges required.

To become familiar with the ServiceNow interface, take this **SELF-PACED TRAINING:** Build My First Application

Note: Delegated development is only available to privately scoped applications.

Build

Once planning is complete, build the data model.

Data

Define the data model first in the build process. Create one or more tables with fields, load the table with demo data, and verify access controls to that data.

Create an app

The first action to take is to either create or open an application record:

- **Create**: Use ServiceNow Studio to create the data model ServiceNow recommends that the Create a custom application option, which allows creates a table, user role, and application menu as the scoped application is created. Any additional tables created will use the application menu and role rather than creating new ones.
- **Open**: If developers do not have the *admin* role, the ServiceNow System Administrator needs to create the application and grant developers a delegated development role. Developers then use Studio to open the application for editing.

Expert tip
The application *Scope* and table *Name*, sometimes referred to as the internal names for these objects, cannot be changed once they are created. However, the application Name and table Label can be changed. Application users only see the internal names in the URL, but internal names should be consistent with what the user sees wherever possible.

Build the Data Model

Create tables and fields on the tables to support the application's data model.

NOTE: ServiceNow automatically adds six fields to each new table. The new fields contain auto-populated information about the table, such as when the record was created, when the record was last updated and by whom, and a unique identifier for the record. These fields cannot be manipulated.

Field name	Database Name	Description
Created by	sys_created_by	The user that created the record

Created	sys_created_on	The date/time which the record was created
Updated by	sys_updated_by	The user who last updated the record
Updated	sys_updated_on	The time at which the last record was updated
Sys ID	sys_id	The unique identifier for the record. This is auto-assigned and unique throughout the instance
Updates	sys_mod_count	A numeric field that counts the number of updates for this record since record creation

New tables can **extend** an existing table to inherit fields and functionality from the table being extended. Add to and modify the components of the extended table to tailor its. The most common table to extend in ServiceNow is the **task** table.

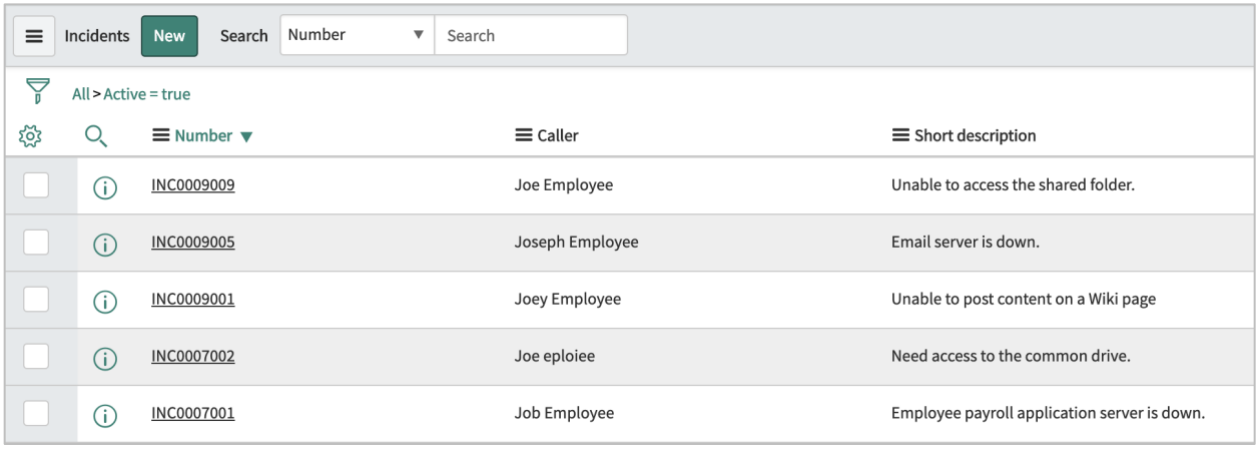
COMMUNITY BLOG: When to Create a New Table vs. When to Extend

DOCUMENTATION: Create a Table

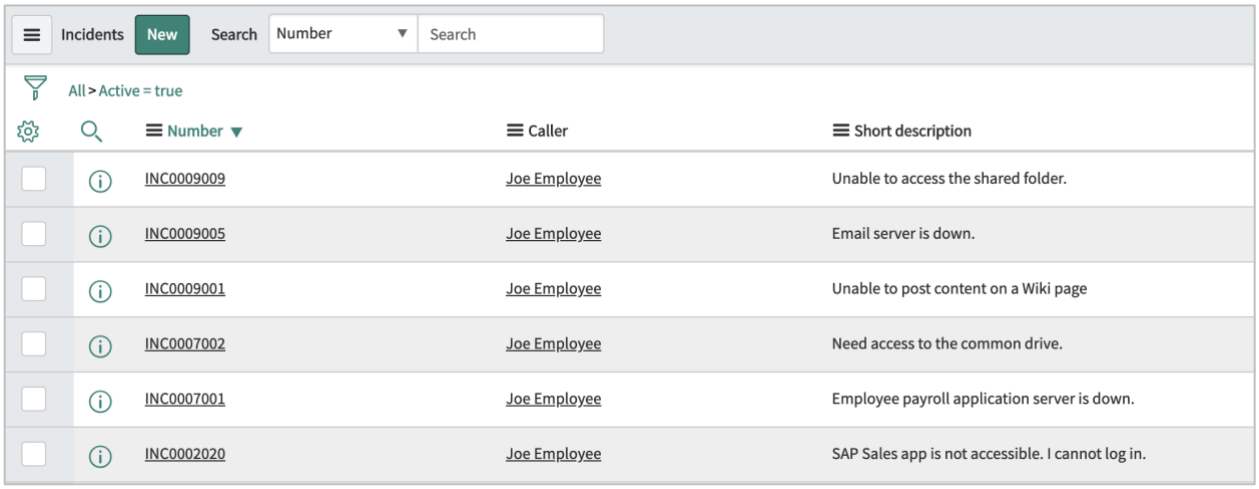
Add fields to the table to support the data model required by the application. ServiceNow has many different field types with built-in validation. Select the field type that best fits that field's data. Plain text (*String*) fields are the easiest to configure, but because users can enter anything, *String* fields can result in bad and inconsistent data that is difficult to use.

EXAMPLE:

What NOT to do: Use a String field for a user's name. Notice the *Caller* field is different for each *Incident* record, but the caller could be the same person:



What to do: Use a reference field type that references the *User* table instead of a *String* field. Users then need to select a single consistent record in the *Caller* field:



Reference fields ensure consistent data by normalizing date in another table in ServiceNow. ServiceNow has over 2000 baseline tables available to reference. Appendix B lists some commonly used tables for building an app.

While a reference field can normalize data, other fields can be used for specific types of data. The ServiceNow Documentation site has the complete list of Field Types. Some common field types:

Field Type	Notes
Integer	Stores number values and can be used in calculations.
Currency	Holds a currency value and will show values in the currency of the logged in user.
Phone number	Includes validation and formatting for E164-compliant phone numbers.
Reference	Displays a record from another table and helps to normalize data.
Choice	Displays a select box with a predefined list of choices. Choice lists should include fewer than ten items.
Date	Stores a date value selected with a date picker. Use Date if you do not need a specific time.
Date/Time	Stores date and time values selected with a date and time picker. Use Date/Time to compare specific times or if the exact time is important.

String	Holds freeform text. Use String if no other field type matches the values stored in the field.

NOTE: Field types should not be changed after a field is created.

Choice Lists vs Reference Fields

Choice lists and Reference fields both offer users a way to choose a value from a list. Choice lists are name/value pairs. Users select from the names, and the field stores the value of the selected choice. Scripts use the value. Add and remove name/value pairs from the choices to manage the list of options.

Reference fields point to a table. Manage choices in the table. The value stored in the Reference field is the *sys_id* of the referenced record.

Choice lists do not require a reference table and are easier to configure than Reference fields. Use Choice lists when the field has ten or fewer options and the options will not change. Consider using a Reference field and table when:

- the field requires more than ten choices.
- the choices will regularly change.
- someone other than an administrator needs to manage the choices.
- the value of the field has an impact on decision logic. For example, Decision Tables in Flow Designer.
- the data has multi-level dependencies between different fields, which would result in complex and unwieldy combinations of choice fields.
- the choices require more than a name/value pair. For example, referencing a *User* record gives the referencing table access to other user details, such as *Email* and *Department*.
- a table already exists that includes the data needed for the field.

When using Reference fields, review the tables available in the instance to reference before creating a table. If creating a new table, check the list of exempt tables in section 2 of the Custom Table Guide. If appropriate, extend the new table from one of these.

Expert tip
Before creating new fields on an extended table, check for an existing field inherited from the base table that has a similar purpose. If a field is found, override the Label on the extended table.

Secure Data

Data security is one of the most important and overlooked aspects of creating an app. ServiceNow automatically configures access control for a new or selected role during the table creation process. Only users with the role can access the table to read, create, write, and delete.

Use Access Control rules to configure table and column-level security in the Now Platform. To properly configure access to an application, developers should understand how Access Controls work and the order in which Access Controls are evaluated. Apply multiple Access Controls together make an Access Control List (ACL).

SELF-PACED TRAINING: Securing Applications

DOCUMENTATION: Access control list rules

When considering security:

- be sure to protect all tables, UI pages, Property pages, and other content with the appropriate Access Controls and roles.
- limit the use of *GlideRecord* queries in Access Control scripts as it can affect performance.

As of the Madrid Platform Subscription model, customers are charged by how many tables a user can access, regardless of whether the user **does** access the table. Configure ACLs to restrict access to a table to ensure that only the users that need access to a table can access the table.

Expert tip
Consider making any auto-populated fields read only. If the system is populating the data a user usually should not be able to do so.

Alternately, secure data on the Now Platform with before-query Business Rules. Before-query Business Rules run before the database query and are limited to controlling read access to a record. Only use before-query Business Rules when necessary. Some considerations when deciding to use Access Controls or before-query Business Rules:

- *GlideRecord* queries will bypass *read* Access Controls on a table but will be restricted by before-query Business Rules on a table.
- When Access Controls restrict read access to records in a list, ServiceNow shows a message saying that access has been restricted for the records. With before-query Business Rules, the number of records in the list total matches the number of records shown to the user. The user receives no indication that some records have been hidden from the list.

Review the *user query* Business Rule on the *User [sys_user]* table for reference.

NOTE: Before-query Business Rules do not take the place of ACLs. Denying users access to a table via before-query Business Rule will still count the table against the subscription model. Use Access Controls to prevent the table from being counted for the users in the Platform Subscription model.

Review the ServiceNow Security Best Practices Guide [requires login] on the HI portal (https://hi.service-now.com) for additional security recommendations.

Encryption

The Now Platform also provides various encryption solutions at the application tier, database tier, and hardware tier. Learn more in the Data Encryption Whitepaper.

Expert tip
Set up security before configuring any interfaces or business logic. Since security affects the data available to interfaces and business logic, waiting until the end of the application build process may cause rework and issues.

Manage Data

With the data model (tables and fields) created and security set up, add data into the application's table(s).

Import Sets

To populate tables in ServiceNow from an external platform, use import sets and transform maps.

To retrieve data from an external source on a regular basis, use a scheduled import.

SELF-PACED TRAINING: Importing Data into ServiceNow

Guidelines for Import Set Performance:

- Break up large amounts of data into smaller sets for faster imports. Consider 100,000 rows to be a baseline and break up anything larger than that into sets of 100,000. For example, importing 10 sets of 100,000 will finish quicker than on set of 1,000,000 records. Also consider using Concurrent Imports with larger numbers of records.
- Importing large data sets simultaneously can put load on an instance. Stagger large imports so the imports do not overlap.
- If possible, deselect the *Run business rules* check box on the transform map table to avoid running Business Rules and other logic during an import. Consider using an *onComplete* transform script to run business logic, such as calculations, at the end of an import rather than on each record as Business Rules do.
- Use default functionality for imports. Whenever possible, avoid writing custom scripts. For example, use the coalesce functionality rather than writing a custom coalesce script.
- Avoid *GlideRecord* queries in an Import Set.
- Make sure any fields configured to *Coalesce* are indexed.

- If replacing a system with a requirement to import historical data, import only the historical data required for the application. Consider storing historical data in a data lake.

Inbound Integrations

In addition to Import Sets, ServiceNow includes APIs to accept data from external platforms.

To push data directly into an application table from another system, use **Web Service Import Sets** instead of writing directly to application tables using the REST Table API or SOAP APIs.

SELF-PACED TRAINING: Web Service Import Sets

To handle data transactions that are more complex than writing data to a table, such as sending an attachment or ordering a catalog item, review the available APIs to see if one exists that supports the required logic.

Use a Scripted REST API or Scripted SOAP Web Service to build REST or SOAP endpoints, respectively. Scripted REST API and Scripted SOAP Web Service endpoints can execute ServiceNow server-side code when the endpoint is consumed by an external system.

SELF-PACED TRAINING: REST Integrations

Design

With the application's data model is created, secured, and populated with data, Create the design elements to access that data.

Primary Interfaces

The primary way for users to interact with your data model is through Forms and Lists or Mobile.

Forms and Lists

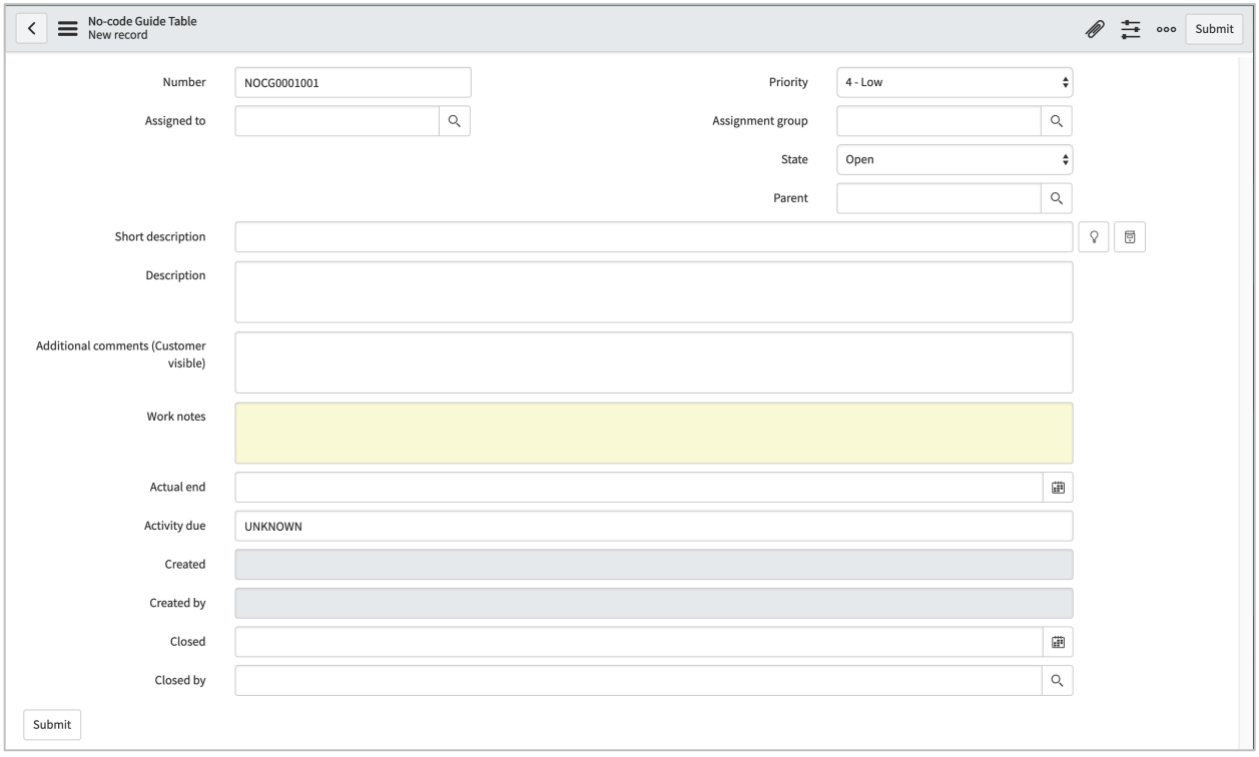
The standard method of accessing data in ServiceNow is through the default forms and lists. A form displays information from one record in a data table, and a list displays a set of records from a table. When configuring forms and lists, follow these guidelines:

1. Keep the number of fields on a form to a minimum. The more fields on a form, the longer the form takes to load resulting in a poor user experience. Use form views to create different sets of fields for different situations.
2. Use form sections to logically group fields together and keep users from having to scroll. The top section of the form should contain the fields that are always needed or used, while the other form sections contain less frequently utilized fields.

3. Make sure fields appear in the right order. For example, the start date field should always come right before an end date field.
4. Use seven or fewer columns in a default list. Users can add more by personalizing their lists.
5. Avoid using a reference field as the first item in the list view as it is shown as hyperlinked text. Clicking on the reference field will redirect the user to the referenced record instead of the list record and results in a poor user experience.

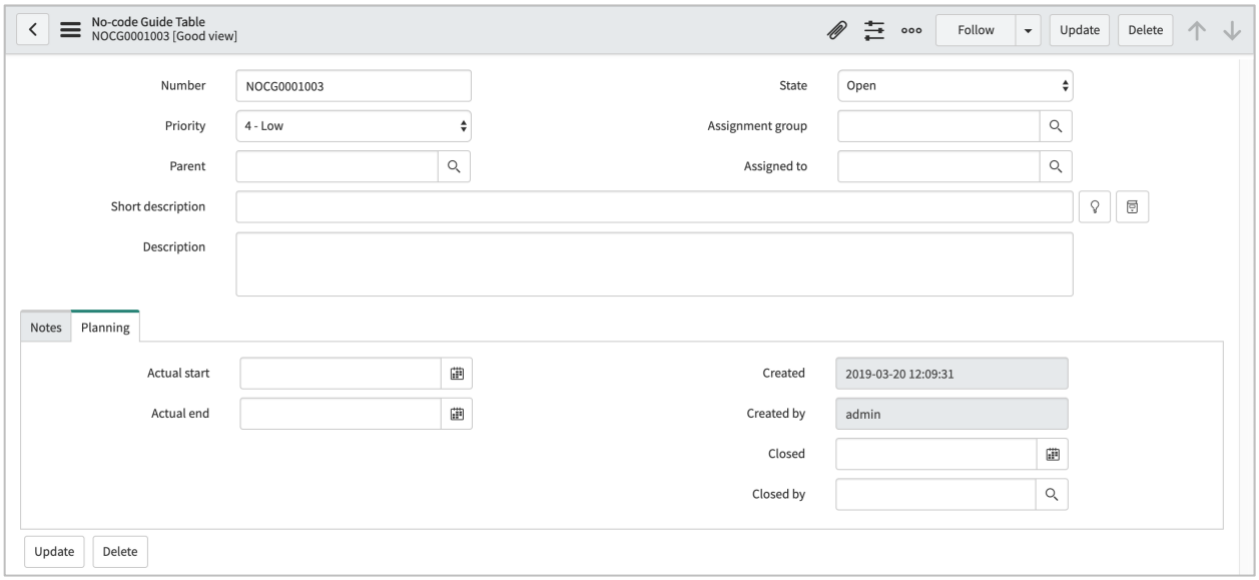
The figure below shows an example of a poorly designed form as it contains the following characteristics:

- The form has no sections. Users need to scroll through the entire form to see all the fields.
- Similar fields are not grouped together. For example, *Assignment group* and *Assigned to* are on different sides of the form.



Below is a well-designed form that contains the following characteristics:

- Fields are grouped together logically, like *Assignment group* and *Assigned to*.
- The form has been broken into sections for easier viewing and data entry.



Mobile

If users will interact with the app on their mobile devices and will need native iOS or Android functionality, like geolocation or offline access to application data, use the ServiceNow Agent app. Use Studio to create a mobile application for the application.

Create applets in the mobile application. Applets are tiles within an application include a master screen, such as a map or a list, and various details screens, such as an activity stream or related lists. Applets should have focused experiences.

Expert tip
Individual applets can be secured by role as well as made available in offline mode.

SELF-PACED TRAINING: Mobile

OTHER RESOURCES: Mobile Resources

Self-Service

Your app also may need a way for end users to be able to access your data model, so there are also some self-service options.

Service Portal

If the application has Requestor or Self-Service users, use Service Portal to provide a friendly web experience.

To give self-service users the ability to easily create application records from the Service Portal, create a record producer. A record producer can provide a better end-user experience than a regular form. Talk to your ServiceNow Administrator about the appropriate catalog and categorization to make the record producer accessible through the Service Portal.

Alternatively, create a Service Portal for an app if the following are true:

- The application needs different branding, navigation, or user experience than an organization's current Service Portal.

OR

- The organization does not have an existing Service Portal,

AND

- The application needs more functionality than the default portals provide.

AND

- The application requires a more customized user experience than the default forms and lists can provide,

OR

- The application needs more control over branding and themes than the default interface provides

Expert tip

Do not try to reuse any existing service portal pages in an app. Create new pages and then reuse components in your pages, like widgets and headers.

Widgets

Widgets are what define the content in the portal. The base system widgets provided with Service Portal can be used, or developers can build custom widgets to fit business needs.

Considerations for creating custom Service Portal widgets for an application:

- Start from an existing widget instead of creating a widget from scratch. To protect existing widgets from accidental modifications, all baseline widgets are read-only.
- When developing a widget, Use the Preview pane to quickly test the widget's behavior. Always test the widget on a portal page before releasing a widget to production.

- Use third-party debugging tools when debugging browser-based applications. For example, the ng-inspector Chrome extension.
- Avoid the use of *$rootScope.$broadcast()*. Instead, use *$rootScope.$emit()* to publish an event to the *rootscope*.
- Use widget options to make widgets more easily reusable. The widget option schema defines the user-configurable fields.
- For field types not supported in the option schema, create an extension table to store a custom widget option schema.
- Make use of Angular Providers, which are reusable components that can be injected into multiple widgets. To ensure quick loading widgets and a high performing portal, create Angular Providers instead of overloading your client controllers with persistent data and additional logic. With Angular Providers, you can maintain data for the lifetime of your Service Portal and reuse components and data objects across multiple widgets.

SELF-PACED TRAINING: Service Portal

Virtual Agent

Consider Virtual Agent, ServiceNow's conversational bot platform. Use ServiceNow Virtual Agent (VA) to build and design bot conversations to help users quickly obtain information, make decisions, and perform common work tasks.

Some considerations when building VA conversation topics:

- If adding VA to a portal, create topics that focus on particular use cases rather than a broad topic.
- Read topics out loud to keep the topic from sounding too robotic or being too verbose.
- Do not try to design the whole interaction at once. Instead take an iterative approach when building the bot and test the conversation frequently.

DOCUMENTATION: Virtual Agent

Communication

With the data model defined and users able to interact with the application, next determine how the application should communicate with users. Configure Notifications to alert users to important application-related events, share application information in the Knowledgebase, and add translations to allow users to interact with the application in their native language.

Notifications

The Now Platform can notify users by email, SMS text message, or push notifications. Build notifications into an application to alert users to important application-related information.

Triggers: Event driven notifications vs table updates:

Configure Notifications to trigger based on updates to a table or based on an event. Use event-based triggering when triggering requirements cannot be easily implemented in the notification conditions or when the notification is triggered from a Workflow. Creating specific events also enables easier notification debugging.

Use the *Notification* activity in a workflow or the *Send an email* action in Flow Designer for simple notification use cases. For complex or critical notifications, trigger an event from a workflow or a Flow Designer flow and configure a notification to fire off that event.

Notification content:

Notifications in ServiceNow support static and dynamic content using Email templates, email scripts, and notification variables.

Notification variables add dynamic information to the body of a notification, like field values from the base record, link to a record, etc. The variables also support dot-walking, which means field values from any of the related records can be included in the content without scripting.

For example, use the *URI_REF* variable to point to the record that originated the email.

Use Email scripts to include dynamic content that is not available in the record or by dot-walking from the base record. Apart from populating dynamic content, use the mail script API to set notification details, such as the recipient and sender addresses, etc.

Note - Scripts entered in the notification body using *<mail_script>* tags may not always work correctly. Always create email scripts at **System Notifications > Email > Notification Email Script** and include them in the notification body with *${mail_script:script_name}*.

Create Email templates for content used in multiple notifications. Adding the content to an Email template enables administrators to create reusable content for the subject line and message body of email notifications.

Configure recipients:

ServiceNow emails can be sent to users, groups, or individual email addresses. When sending to groups, check the *Include members* field on the group record for the notification to be sent to all members of the group in addition to the group email.

Subscription-based notifications - Select the *Subscribable* option on the notification to allow recipients to pick and choose the emails they want to receive.

For a subscription-based notification, the *Mandatory* option can be set to *true* for the recipients to receive the notification regardless of their individual preferences. Optionally, configure unsubscribe links in the outgoing email to allow users to remove themselves from the notification.

ServiceNow uses email watermarks to correctly process user responses to notifications. Email notifications automatically include watermarks unless the *Omit watermark* option is checked in the notification. Omit watermarks only when no email response is expected from the recipients.

The following KB article provide troubleshooting steps for the most common notification issues.

https://hi.service-now.com/kb_view.do?sysparm_article=KB0521382

Translations

When using one of the Internationalization plugins, most of the fields in the instance are automatically translated. However, customizations are not translated automatically, and need to be translated by hand. In this case, it is best to locate the individual untranslated strings, and insert those translations manually.

DOCUMENTATION: Language Internationalization Support

To implement a new language, activate the plugin for the new language. Then export the existing values from any of the Translation tables.

For example, export all *sys_choice* records. Provide the export to a translator to provide translations for the *Label* field in the provided file. The *Value* field should not change. Update the *Language* value to the new language. Then, upload the translated values back to the *sys_choice* table.

Note: To insert values, make sure you are setting all the necessary fields; table, element, language, label, value, and sequence (as well as Dependent value and Hint where applicable). This logic is the same for all translation tables.

Reporting and Dashboards

Most applications will have some level of reporting requirements. Reports should be actionable to drive change. Follow these guidelines when building reports:

- Reporting on large tables can have performance impact on a ServiceNow instance. Be sure to filter by a date range or other limiting criteria rather than showing all records on the table.
- Grouping records by fields that contain many possible values can negatively impact performance.
- If running a report gives *Long running transaction timer* message, consider adding more data filters to reduce the report run time.
- Schedule reports needed on a regular basis to be sent via email.

Use dashboards to show multiple reports on one page. Be careful with the number of reports on a dashboard. Too many reports on a dashboard and multiple users are using that dashboard can negatively affect instance performance.

SELF-PACED TRAINING: Reporting and Analytics

Logic

The next step in designing an application is to build logic. Logic includes form logic (what people can and cannot see/use on a form) and business logic (rules that govern what happens to data when it is entered).

Scripting and Modifications

Before writing any code, be aware of the impact on upgrades and the adoption of new ServiceNow features. Particular care should be taken when modifying baseline artifacts and processes.

Consider the following before scripting:

- Asses the requirement. Is the logic critical to the functioning of the app?
- Determine if ServiceNow can be configured to fulfill the requirement without code.
- Leverage options, such as Flow Designer, Virtual Agent, and UI Policies to take advantage of platform capabilities without writing code.
- Low and no-code approaches to logic are easier to debug and upgrade.

Examples of when scripting is appropriate:

- Building Flow Designer actions
- Creating a Scripted REST API
- Creating logic for scoped applications in Script Includes
- Customizing and creating widgets for Service Portal

Evaluate the business requirement and consider a no-code route before using a scripted solution.

Be aware of ServiceNow enhancements. For example, in the Madrid release, Virtual Agent conversations have more no-code options than London. Read release notes and other publications. Get certified and stay current with your certifications.

To better understand when to customize, review the Innovate at Scale Success Playbook on the Customer Success Center.

Modifying Default Behavior

In the past, one of the strategies used was to copy the artifact to update and to deactivate the original. The copy/deactivate approach is no longer recommended due to the following issues:

- Developers cannot tell if a deactivated artifact was upgraded without research.
- Two files, the original and the copy, need to be maintained. Maintenance doubles each time a customization is made.
- With each release, the customized record becomes older.
  - Customers do not receive the enhancements included in a new release.
  - A new release may rely on the original record being updated.
  - Developers may make more changes to compensate for the original record being inactive.

A script where only the *Active* flag is changed will be updated, but the script does not appear on the skipped list. With the copy and deactivate strategy, a developer has less visibility into customizations and cannot easily assess or revert to the baseline version.

Rather than copying and deactivating the original artifact, edit the artifact directly. The ServiceNow Upgrade Engine will add the latest version to the version history and report that the artifact was skipped. Developers can see a new version is available with the upgrade.

Form Logic

Controlling what users see when they visit a form can greatly increase productivity and responsiveness. For example, users should only see fields that are useful to them. Users may only need to see certain fields based on what is configured on the form. Apply form logic to control what is visible, read-only, and mandatory on a form.

The following question will help direct you to the right decision for when to control user access to information: Is this a suggestion or enforcement? A suggestion makes the form easier to complete whereas enforcement forces the user to do something in order to complete the form.

UI Policies are useful for conditional *suggestions* like showing and hiding fields or adding field messages based on another field's value, while Data Policies and Business Rules are better suited for doing conditional *enforcement* like making a field mandatory.

The best user experience is to utilize both suggestion and enforcement together.

SELF-PACED TRAINING: UI Policy Section

Build UI Policies and Data Policies to handle client-side activities before scripting any client-side logic. Use of Client Scripts to validate user input and provide feedback while the user is completing the form.

Some general practices for client scripting are:

- Optimize for performance by using asynchronous *GlideAjax* over client-side *GlideRecord* or multiple *getReference()* calls.
- Keep the *isLoading* check in *onChange* client scripts.
- Keep the *newValue* check and add a *newValue != oldValue* check.
- Use all client-side scripts possible before making a server call with *GlideAjax*. Server roundtrips can impact performance.

Some client scripting practices to avoid are:

- Global Client Scripts or Global UI Scripts: Global scripts will run on every page load and introduce browser load delay.
- DOM Manipulation: Using document object model manipulation against default UI elements introduces upgrade risk and maintainability issues. The exception is using DOM manipulation against the DOM in pages authored in the same scoped application, like UI Pages or Service Portal widgets.

RESOURCE: Client Scripting Technical Best Practices

Business Rules and Script Includes

Business Rules are server-side actions that can be run during CRUD (Create, Read, Update, Delete) operations on instance records. Some good practices when using Business Rules are:

- Keep Business Rules small and specific.
- Avoid modifying base system Business Rules.
- Use Script Includes instead of global Business Rules.
- Use scripting only when necessary.
- Store reusable script logic in a Script Include.
- Use queries to limit records processed within a Business rule.
- Avoid client-callable Business Rules to improve efficiency when running client scripts.
- Always use a condition with Business Rules to control when the Business Rule runs. Running Business Rules with conditions can also aid in debugging. Business Rules rarely run with no conditions.

Business Rules can be configured to run before or after a database operation. They can also be configured to run asynchronously and also before displaying a form or executing a query.

Value	Runs	When to use	Example

Before	Synchronously before the database operation	• Set or update values on the current object as part of the save operation • Validate and abort execution if required	A developer wants to set the state of the current record based on another input in that record.
After	Synchronously after the database operation	• Trigger events and notifications after the database update to access the previous object or to make something occur in sequence • Update related records other than the base table being updated to access the previous object or to make something occur in sequence	A developer wants to cascade values from the current record down to child records.
Async	Asynchronously executed as a separate process after the database operation is completed	• The process triggered by the rule may take a while to run • When the user who triggered the operation does not need the output right away • Trigger events, notifications, or related record updates when access to the previous values of the record or a specific sequence of actions is not required	A developer needs to trigger an external process that may take a while or update a large number of records.
Display	Executed every time the corresponding form is displayed	• Used to make server-side objects available to client-side scripts	A developer wants to write information about a user associated with the current record to the g_scratchpad object to use in a client-side script.

Note: *current.update()* should not be used in any Business Rules. Using *current.update()* triggers an additional database operation, which could cause duplicate notifications, recursive loops, etc.

RESOURCE: Business Rules Technical Best Practices

Use Script Includes to store JavaScript functions and classes for use by server scripts. Each Script Include defines either an object class or a function that can be reused among any server-side scripts.

Store any code that might need to be used elsewhere in a Script Include. Call the Script Include from a Business Rule, UI Action, workflow script, Scripted REST API, etc. Instead of calling a Business Rule from a UI Action or a UI Action from a Scripted REST API, put the code in a Script Include and call the Script Include from both places.

Keeping functions in a Script Include allows testing of the function before deploying the function in other scripted areas, thus reducing overall development and testing time.

DOCUMENTATION: Business Rules

Flow Designer and IntegrationHub

Flow Designer is a Now Platform feature that enables rich process automation capabilities in a consolidated design environment. Flow Designer enables process owners to use natural language to automate approvals, tasks, notifications, and record operations without having to code.

Flow Designer vs. Workflow

For any new process flow requirements, ServiceNow recommends using Flow Designer over the legacy workflow for almost all circumstances except for the few functions that are only available in the workflow tool.

Flow Designer vs Business Rules

You should use Flow Designer instead of Business Rules unless:

- Business logic needs to run in a specific sequence with other Business Rules. For example, new business logic needs to run after one Business Rule but before another.
- Logic needs to execute immediately before or after writing to the database in the same thread.
- The logic only calls a Script Include.

When designing a flow, follow these design principles:

- **Single Purpose**: each flow should have a singular goal.
- **Reusability**: Design with reusable subflows in mind (approval is a great example)
- **Clarity**: The language and layout of a flow should make each action's purpose clear.

Start with a white board design of a business flow. Then build the flow action by action to align with the process. More than one flow may be required for a for a single process to keep to the design principles.

Use the following practices when working with **Flow Designer**:

- Use records, not SysIDs.
- Provide a guided experience with inline documentation.
- Learn how to use template objects to work with both static and dynamic inputs.
- Avoid passing around blobs of data unless absolutely necessary.
- Only pass information to a flow that the flow is going to use.

Use the following practices when working with Flow Designer **Actions**:

- Always create actions under the scope of the application's spoke, if applicable.
- Set access to **Accessible from all scopes** in actions to be able to reuse actions across other apps and scopes in the future.
- Set *Protection* to **Read-only** to avoid any unwanted edits to the actions by users.
- Make sure inputs have a specific type.
- Ensure that **Mandatory** is selected where required.
- If using a **choice** input type, use a default value.

Use the following practices when working with **IntegrationHub**:

- Create one spoke per integration system. Only put actions for a single system in a spoke.
- When creating the scoped app for the spoke, use a version naming convention that makes sense.
- Use a connection alias instead of an inline connection. The Base URL will be automatically extracted.
- Use connection attributes under the Alias to pass the version in a REST step giving future flexibility for versioning in the resource path.
- Use *Save as Attachment* to save the content in the response instead of creating another step to save the data.
- If the Alias is dynamic, make Alias one of the inputs and use the data pill to provide the Alias.

Use the following practices in Flow Designer and IntegrationHub for **Error Handling**:

- Create a Script Include to handle errors.
- Write short and understandable error messages.
- Incorporate all of the possible error messages the API returns.
- Ensure that the outputs from the integration step are validated before using them.
- Fail Early: If the inputs are not available, do not call the integration.

SELF-PACED TRAINING: Flow Designer

SELF-PACED TRAINING: IntegrationHub

Validate

As the application is built, validate that it works as expected.

Unit Testing

Unit/Story testing ensures requirements specified in a story are validated before closing the story. A Story/Unit is a smallest testable portion of system or application that can be configured and executed.

When the configuration of the story is complete, developers need to unit test the features not only in the context of that particular story, but also other related stories that share components with the current story.

As a good practice, developers need to assign the story to process owner or designated stakeholder to validate the story configuration meets expected outcomes before closing the story.

ServiceNow's Automated Test Framework (ATF) is primarily meant for automating functional testing of applications but in few cases can be used to automate unit testing of configurations that involve Script Includes and Business Rules.

System Testing

System testing is performed on a complete system when development is completed. Test the overall interaction of components and integrations with other applications within scope. System testing is performed by the QA/Testing team, but developers need to collaborate with the QA team and process owners to ensure test cases provide comprehensive coverage. Developers will be responsible for remediation of issues found during System Testing.

Automated Test Framework

Automated Test Framework (ATF) should be leveraged for automating functional system testing of ServiceNow applications to reduce testing time and costs and make testing repeatable and UI independent. When creating test cases, follow these guidelines.

When Creating Tests:

- Use parameterized testing to avoid duplicate test cases.
- Follow a Test naming standard.
  - *<app initial>: <functionality that is being tested>*
    CSM: Resolve case
- Describe each test's use case in its description.
  - Description: "Sample that tests <x> use case"
- Develop tests on a Dev instance and promote/run the test on a Test instance.

- Clones wipe out tests. Use one of these options to preserve tests:
  - Bundle tests in a scoped app and upload the app to GIT.
  - Save tests before the clone.
  - Promote tests to prod instance, but **DO NOT EXECUTE THE TESTS IN PROD.**
- Create self-contained tests.
- Create new server-side or REST test steps any test steps are missing.
  - Ex: Email body verification.
- Use server-side test step whenever possible and when screenshots are not important.
- Start with the *Impersonate* step.
- Be aware of browser throttling.
- Use the Test Logs and Test Transactions to troubleshoot test errors.

When Creating Test Suites:

- Follow a test suite naming standard.
  - <app initial>: <functionality that is being bundled>
    Ex: ITSM INT: Use cases
- Describe the suite.
  - Test suite description: "This is a sample test suite to test <x> plugin/application."
  - Provide any additional information possible in the description.
- Organize test suites by feature areas.

User Acceptance Testing

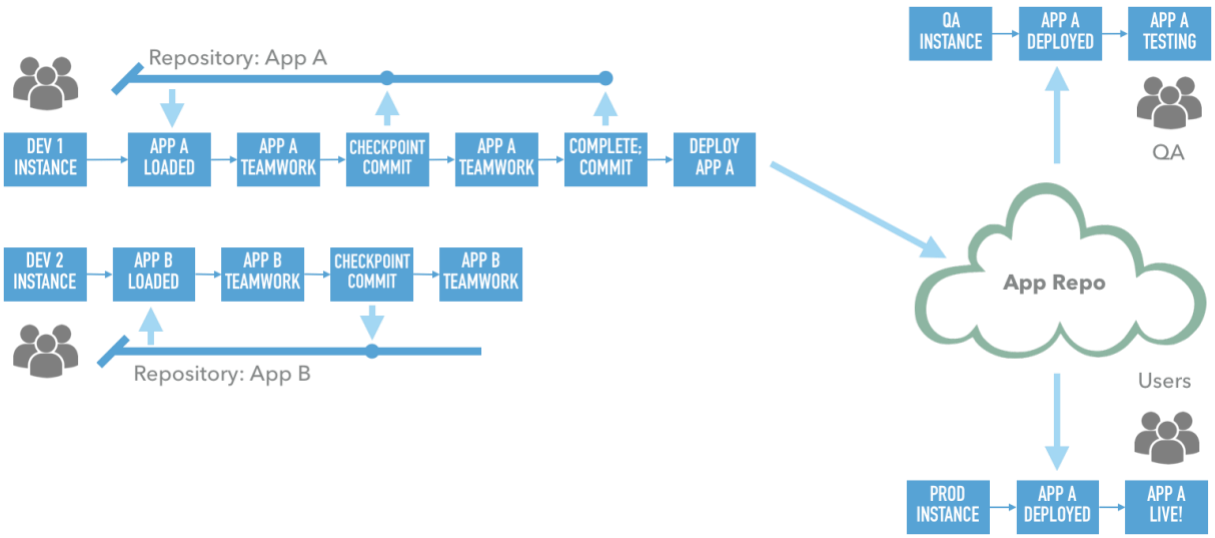
User Acceptance Testing (UAT) is a test conducted to evaluate the application's compliance with the business requirements and assess whether the application is acceptable for delivery. Users, customers, or other authorized stakeholders perform acceptance testing. Developers will be responsible for remediation of issues found during System Testing.

Deploy

Once the application is built and validated, the application needs to be moved to the production environment. Applications can be moved through an application repository or by using Update Sets. Applications should be deployed to test environments prior to moving to production.

Application Repository

Publishing an application to the App Repo makes this version of the application available to all of an organization's ServiceNow instances. Use the App Repo to deploy an application to QA / Test instances (for testing) and finally to Production (Prod) instances.

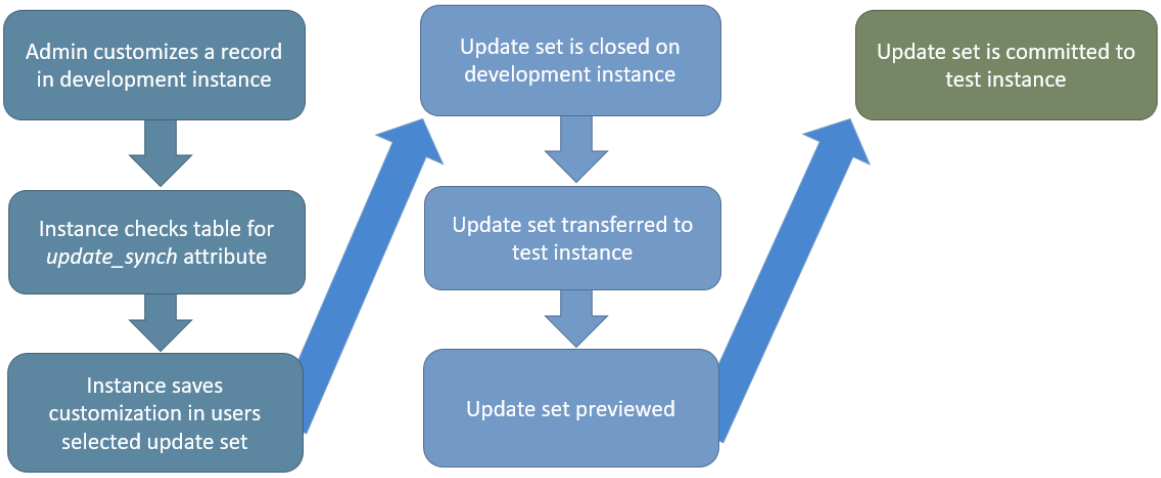


Documentation: Publishing an application to the application repo, Install an application

Update Sets

If the application repository cannot be used to deploy applications, use Update Sets instead.

The diagram shows the best practice lifecycle of an Update Set to deploy a customization from the development instance to the test instance.



Practices that lead to a quality development and release process:

- Always move customizations from the bottom of the stack up.
  - Ensures down-stack instances match up-stack instances.
  - Customizations introduced mid-stack can be overwritten by future pushes from down-stack.
  - Common scenarios include:
    - Fixes need in test or prod – always push them from dev up.
    - Common Prod admin customization, such as choice lists – always push updates from dev up.
- Always review updates contained in an Update Set before transferring.
  - Look for updates associated with other dev efforts and updates associated with testing.
  - Watch for system properties and integration end-points changes.
    Ex: pushing sys_properties change that directs all email to test email account
  - Move updates to a "scrap" Update Set rather than deleting the update.
- Always test after pushing.
  - Ensure that all desired customizations are captured and applied as expected.
- In situations with multiple, parallel releases, ensure communication and coordination between the development teams.
- Avoid experimenting on the development instance.
  - Customizations can be captured accidentally and migrated by other team members.
- Do not capture development in the *Default* Update Set.

Naming Convention

List all user Story numbers with a short description in the *Description* field of an Update Set. Include all manual steps that are required to deploy the Update Set.

Some typical examples of manual steps that are needed for a deployment that are not captured in an Update Set:

- Plugin activation.
- Transfer of tables that are not tracked in the Update Set (typically, starting with "x_" or "u_").
- Creation of database indexes on the tables. The index creation is not tracked via Update Set and needs to be done manually.

Update Set Management

Be sure the correct Update Set is selected when working on a story or a defect and check the records in the Update Set daily.

Do not manually move *sys_update_xml* records between Update Sets! The only exception is to move a record to the *Default* Update Set.

Update Sets capture configuration information but not task or process data. For example, Update Sets track Service Catalog item definitions and related configuration data like variables and variable choices. However, the orders (requests, items, catalog tasks) placed in testing are not tracked by Update Sets.

Be aware of Update Set DOs and DON'Ts:

- To remove a specific *sys_update_xml* record from the current Update Set, move the record to the *Default* Update Set and populate the field *sys_update_set.comments* of the record with reason for moving the record to the *Default* Update Set.
- Never move customization records from one Update Set to another Update Set.
- Never delete an Update Set unless the Update Set has been merged successfully into a new Update Set.
- Always use data extracts or Import Sets to move data from one instance to other (and not Update Sets).

Update Set Batching

Batch Update Sets to preview and commit Update Sets in bulk.

Dealing with multiple Update Sets can lead to problems, including committing Update Sets in the wrong order or inadvertently leaving out one or more sets. Avoid these problems by grouping completed Update Sets into a batch.

The system organizes Update Set batches into a hierarchy. One Update Set can act as the parent for multiple child Update Sets. A given set can be both a child and parent, enabling multiple-level hierarchies. One Update Set at the top level of the hierarchy acts as the base Update Set.

Previewing or committing the base Update Set previews or commits the entire batch. The system determines the processing order and checks for collisions based on the dates the changes were recorded and on their sequential ancestry. Their ancestries are the specific instances in which the changes in the Update Sets took place.

Update set batching can be applied to releases, where an empty parent Update Set is created for the release, and actual Update Sets are included in the release as children.

Advantages of using Update Set batching are:

- Individual Update Sets can be removed from the release at the last moment.
- Batching is similar to merging, except batching allows updates to be removed.
- Batch Update Sets are easy to deploy. Only the Parent Update Set needs to be processed.

DOCUMENTATION: System Update Sets

Conclusion

Congratulations, the application has been planned, built, validated, and deployed to solve the business problem it was designed to address.

What to do next

Now that the app is deployed, think about how to improve and enhance it. Here are some suggestions for determining where to go next:

- The people using the application day-to-day will be the best source of feedback. Talk to them about what new features or changes they would like to see.
- Determine if additional related process flows can be automated through Flow Designer.
- Determine if new IntegrationHub spokes can be leveraged for new integrations.

Feedback

For any feedback or questions regarding this document, please email platformenablement@service-now.com or leave feedback on the developer site.

APPENDIX A – Now Platform Development Tools

Studio

ServiceNow Studio is an Integrated Development Environment (IDE) that offers ServiceNow application developers a centralized interface to create and manage their applications. Studio is the primary interface for creating and updating an application. Use Studio for source control integration, delegated development, or publishing an app to an Update Set or to the application repository.

- How to access studio
- Practice with Studio

Import Sets

Import sets allow administrators to import data from various data sources, and then map that data into ServiceNow tables.

- Intro to Import Sets
- Practice with Import Sets

REST API Explorer

The REST API Explorer uses information from an instance to provide a list of endpoints, methods, and variables that developers can use to build and send a REST request. After building the request, the REST API Explorer provides code samples in multiple programming languages that developers can use to send the request, along with detailed request and response information.

- Get Started with REST API Explorer
- Practice with the REST API Explorer

Form Designer

Form configuration involves changing the form layout. The form layout changes what appears on the form and the related list layout changes which related lists appear at the bottom of the form. With Form Designer you can change aspects of a form such as showing or hiding fields and adding sections and annotations.

- Using the Form Designer
- Practice with Form Designer

Mobile

The ServiceNow mobile app is built with a mobile-first design. Configure the ServiceNow mobile app so users can access an instance on a tablet or smartphone. Configuration for the mobile application takes place in Studio.

- Mobile in Studio
- Practice with Mobile

Service Portal

- Service Portal Configuration
- Practice with Service Portal

Virtual Agent

Consider adding a conversational bot to Service Portal or third-party messaging platforms that can chat with users and take actions based on these chats. Specify inputs and take action on those inputs, like creating or looking up records. Find out more about Virtual Agent in the ServiceNow docs site.

- Getting Started with Virtual Agent
- Practice with Virtual Agent

Reporting

ServiceNow Reporting enables users and developers to create and distribute reports that show the current state of instance data, such as the number of open tickets of each priority. Reporting functionality is available by default for all tables, except for system tables.

- Getting Started with Reports
- Practice with Reporting
- Practice with Performance Analytics

Flow Designer

Flow Designer is a Now Platform feature that enables rich process automation capabilities in a consolidated design environment. Flow Designers enables process owners to use natural language to automate approvals, tasks, notifications, and record operations without having to code.

- Getting Started with Flows
- Practice with Flow Designer

IntegrationHub

IntegrationHub enables execution of third-party APIs as a part of a flow when a specific event occurs in ServiceNow. These integrations, referred to as spokes, are easy to configure and enable you to quickly add powerful actions without the need to write a script.

- Practice with IntegrationHub

Script Debugger

The Script Debugger allows application developers to debug any server-side JavaScript that runs in an interactive transaction, such as Business Rules, Script Includes, Script Actions, or UI Actions that require a response in order to proceed.

- Get started with the Script Debugger
- Practice with the Script Debugger

Syntax Editor

The syntax editor provides support for editing JavaScript scripts.

- Intro to Syntax Editor

Automated Test Framework

With Automated Test Framework, create and run automated tests to confirm an application works after making a change, such as after an upgrade, during application development, or when deploying instance configurations with Update Sets. Use test results to identify changes needing review.

- Get started with ATF
- Practice with ATF
- Self-paced ATF Fundamentals Training

APPENDIX B – Now Platform Resources

Onboarding a New Developer

If you are new to ServiceNow development, how should you get started? This section will provide some options for developers who are new to ServiceNow to ramp up on the Now Platform.

Training:

- New to ServiceNow Learning Plan – Self-paced | Free
- Learn JavaScript on the Now Platform – Virtual | Free
- Scripting in ServiceNow Fundamentals Course – In-person/Virtual | Paid
- Application Development Fundamentals Course – In-person/Virtual | Paid

Other things to do:

- Join a ServiceNow User Group
- Join a Developer Meetup
- Attend the Knowledge Conference
- Try building an app!

Resource Links

Here are some resources you should bookmark for help.

- Product Documentation – Documentation on all of ServiceNow's features and functionality
- Developer Portal – Portal where developers can get a developer instance and train
- Developer Training – Specific, constantly updated self-paced training classes
- Community – Active community forum where you can ask questions
- Customer Success – Best practice information
- Training Courses – ServiceNow's virtual or in-person training classes

Expert tip

One of the best resources for your application is existing applications. You can look in your instance and see how others have built and structured their application and learn from their example. Or look through the ServiceNow Share site to find applications others have posted.

APPENDIX C – Common Tables within ServiceNow

Label	Name	Description
User	sys_user	All ServiceNow instance users
Location	cmn_location	List of all user locations. Users are typically associated with a location.
Group	sys_user_group	List of all of the groups. Users are typically associated with groups and inherit any security roles associated with the groups.
Company	core_company	List of companies that interact with your organization.
Role	sys_user_role	List of security roles in the instance. Some will be default roles and some will be created by your organization.
Task	task	The most common base table to extend. Task has fields and functionality related to assigning work across teams and individuals, managing the state or the task, and other functions.