

# TITOLO DEL DOCUMENTO

December 8, 2025

## INDICE

1	Introduction to ServiceNow Scripting	6
2	ServiceNow APIs	6
3	ServiceNow Script Editor	7
3.1	Controllo Sintassi in Tempo Reale	7
3.2	Toolbar del Script Editor	7
3.3	Colorazione del Codice	7
3.4	Click Destro e Navigazione	8
4	Introduction to ServiceNow Scripting: Module Recap	8
4.1	Concetti Chiave	8
5	Preparing Data for Import	8
5.1	Esempio Pratico	9
5.2	Mappatura delle Colonne	9
6	Data Sources	9
6.1	Tipi di Data Sources	9
6.2	Creazione di Data Sources in Applicazioni Scoped	9
6.3	Esempio: JDBC per MySQL	10
7	Data Import Process	10
7.1	Utilizzo di Studio	10
8	Load Data	11
9	Create a Transform Map	11
9.1	Configurazione del Transform Map	12
9.2	Mappatura dei Campi	12
9.3	Verifica dei Dati Mappati	13
10	Running a Transform Map	13
10.1	Passaggi principali	13
10.2	Nota Importante	14
11	Verify the Data Integrity	15
11.1	Aprire l'Import Set	15
11.2	Controllare Import Set Runs	15
11.3	Controllare Import Set Rows	16
11.4	Gestione dei Campi Obbligatori	16
12	Importing Data from NeedItImportData.csv	16
12.1	Preparazione del File CSV	16
12.2	Caricamento nella Staging Table	17
12.3	Creazione e Configurazione della Transform Map	17
12.4	Esecuzione della Transform Map	17
12.5	Gestione degli Errori	18
12.6	Ripetizione dell'Import e Verifica Finale	18
12.7	Punti Chiave	19
12.8	Verifica Finale dei Record Importati	19









## 1 Introduction to ServiceNow Scripting

ServiceNow utilizza JavaScript standard per estendere le funzionalità delle applicazioni. Le API di ServiceNow (*Application Programming Interfaces*) forniscono classi e metodi per:

- Interagire con tabelle del database: interrogare, aggiornare, creare, cancellare record
- Validare dati
- Scrivere su file di log
- Mostrare alert, conferme o messaggi agli utenti
- Avviare workflow, job schedulati ed eventi
- Interagire con servizi web di terze parti
- E molto altro

Il motore JavaScript di ServiceNow usa lo standard ECMAScript 5 a partire dal rilascio *Helsinki*. Per supportare sia script esistenti sia nuovi script sviluppati secondo lo standard ES5, il motore JavaScript opera in due modalità:

- **Compatibility Mode:** per script globali e rilasci precedenti a Helsinki
- **ES5 Standards Mode:** per script delle applicazioni scoped

Il motore determina dinamicamente quale modalità usare per ogni script in base allo scope e alla data di creazione dello script.

**Nota:** a partire dal rilascio *Tokyo*, ServiceNow supporta ECMAScript 2021. Maggiori dettagli si trovano nel blog *Tokyo Introduces ECMAScript 2021*.

## 2 ServiceNow APIs

Le API di ServiceNow permettono di interagire con la piattaforma sia dal lato client sia server, e anche tramite applicazioni esterne usando REST. Le principali categorie di API sono:

- **Client:** API lato client per applicazioni desktop
- **Client Mobile:** API per la vecchia applicazione mobile ServiceNow Classic (non per ServiceNow Agent, Now Mobile o Onboarding)
- **Now Experience UI Framework:** API per componenti dell'Agent Workspace
- **Server Scoped:** API lato server per applicazioni scoped
- **Server Global:** API lato server per applicazioni global
- **REST:** API REST per interagire con un'istanza ServiceNow

La documentazione completa delle API è disponibile sul *ServiceNow Developer Site*. Nel Reference menu, selezionando una API, si possono consultare:

- Nome del metodo
- Descrizione del metodo
- Tipo di dato restituito
- Descrizione del dato restituito
- Esempio di script

Tutti i contenuti delle API sono versionati. Quando si consulta la documentazione, usare il selettore del release per scegliere la versione corretta. Applicazioni sviluppate in scope personalizzati non usano le stesse API server-side delle applicazioni globali. Per selezionare lo scope corretto usare il tab *Scoped (custom)* o *Legacy (global)* nel navigatore della documentazione.

È buona pratica consultare la documentazione API frequentemente durante lo sviluppo di applicazioni sulla piattaforma Now.

enddocument

## 3 ServiceNow Script Editor

Tutti gli script in ServiceNow, indipendentemente dal tipo, hanno due componenti principali:

- **Configurazione:** specifica quando eseguire la logica dello script
- **Script:** contiene la logica che viene eseguita quando le condizioni di configurazione sono soddisfatte

Sebbene la configurazione vari a seconda del tipo di script, il *Script Editor* è lo stesso per tutti gli script. Le principali funzionalità includono:

### 3.1 Controllo Sintassi in Tempo Reale

---

Il controllo sintassi individua errori JavaScript mostrando indicatori accanto ai numeri di riga:

- Avvertenze: cerchio giallo con punto esclamativo
- Errori: cerchio rosso con X

Passando il mouse sugli indicatori si ottengono maggiori informazioni. La verifica riguarda solo errori di sintassi (ad esempio punto e virgola mancante, parentesi non chiuse), non errori logici o nomi di variabili/metodi.

### 3.2 Toolbar del Script Editor

---

La toolbar include comandi per:

- Aiuto del Script Editor
- Abilitare/disabilitare l'evidenziazione della sintassi
- Commentare/decommentare righe
- Formattare il codice (indentazione)
- Sostituire una stringa o tutte le occorrenze
- Cercare stringhe, navigare tra occorrenze
- Modalità schermo intero
- Visualizzare scorciatoie da tastiera
- Salvare il record
- Abilitare/disabilitare il controllo sintassi in tempo reale
- Aprire il debugger lato server (non disponibile per script lato client)

### 3.3 Colorazione del Codice

---

Lo Script Editor applica colorazione automatica per migliorare la leggibilità:

- Verde: commenti
- Magenta: oggetti JavaScript
- Blu: stringhe e parole riservate

La palette dei colori non è configurabile dall'utente.

## 3.4 Click Destro e Navigazione

---

Cliccando con il tasto destro su testo o oggetti si possono accedere rapidamente a:

- Documentazione API
- Definizioni dei record
- Lista dei record
- Altri script relativi allo stesso elemento

## 4 Introduction to ServiceNow Scripting: Module Recap

### 4.1 Concetti Chiave

---

- Lo scripting permette di aggiungere e estendere funzionalità nelle applicazioni
- La documentazione ufficiale delle API su *developer.servicenow.com* è la fonte principale di informazioni
- Il *Script Editor* viene utilizzato per scrivere la logica di tutti i tipi di script
- Il controllore della sintassi individua solo errori di sintassi JavaScript, non tutti gli errori possibili
- Esistono API per script lato client, lato server e REST
- Utilizzare il selettore della versione per scegliere il rilascio corretto
- Per le API server-side, selezionare lo *scope* appropriato: Scoped o Global

## 5 Preparing Data for Import

Prima di importare dati in ServiceNow, è consigliabile seguire alcuni passaggi chiave:

- Comprendere quali dati si stanno importando
- Decidere come trattare dati incompleti o errati
- Creare un piano per mappare le colonne del file sorgente ai campi della tabella target

	A	B	C
1	Employee ▼	Day type ▼	Date ▼
2	Edna McPhearson	birthday	3/3/83
3	<del>Edna McPhearson</del>		<del>3/3/83</del>
4	Edna McPhearson	work anniversary	1/14/11
5	Mabel Gerkowski	work anniversary	12/18/16
6	Mabel Gerkowski	birthday	11/11/95
7	Myrtle Hadley	birthday	3/20/68
8	Myrtle Hadley	work anniversary	3/20/13
9	Opal Northcott	birthday	1/1/91
10	<del>Opal Northcott</del>	<del>birthday</del>	<del>1/1/91</del>
11	Opal Northcott	work anniversary	2/2/12

È molto più difficile rimuovere dati indesiderati dopo l'importazione che pianificare correttamente in anticipo.



## 5.1 Esempio Pratico

Storicamente, compleanni e anniversari dei dipendenti erano tracciati in un foglio di calcolo. Per i nuovi dipendenti, le occasioni speciali vengono gestite nell'app *Employee Special Days*. Per centralizzare i dati, i record storici verranno importati nella stessa applicazione, eliminando prima i record duplicati o incompleti dal file sorgente.

A	B	C
Employee	Day type	Date
Edna McPhearson	birthday	3/3/83
Edna McPhearson	work anniversary	1/14/11
Mabel Gerkowski	work anniversary	12/18/16
Mabel Gerkowski	birthday	11/11/95
Myrtle Hadley	birthday	3/20/68
Myrtle Hadley	work anniversary	3/20/13
Opal Northcott	birthday	1/1/91
Opal Northcott	work anniversary	2/2/12

Occasions OCC03001		Update	
Number	OCC03001		
* Employee	Alissa Mountjoy	* Special occasion	Birthday
* Employee email	alissa.mountjoy@example.com	* Occasion date	1997-01-08
		Active	<input type="checkbox"/>

## 5.2 Mappatura delle Colonne

Creare un piano che associ ogni colonna del file sorgente ai campi della tabella target. Ad esempio:

- Colonna **Employee** → campo **Employee**
- Colonna **Day type** → campo **Special occasion**
- Colonna **Date** → campo **Occasion date**

Questa mappatura garantisce coerenza dei dati e riduce gli errori durante l'importazione.

## 6 Data Sources

Le Data Sources definiscono quali dati devono essere importati in ServiceNow. Solo gli utenti amministratori possono creare Data Sources.

### 6.1 Tipi di Data Sources

Alcuni tipi di Data Sources comuni includono:

- Microsoft Excel
- CSV
- JDBC
- FTP
- HTTP
- XML

### 6.2 Creazione di Data Sources in Applicazioni Scoped

Per creare una Data Source in un'applicazione scoped, utilizzare *Studio*. Per esempi di configurazioni, aprire: *All menu > System Import Sets > Administration > Data Sources*.

## 6.3 Esempio: JDBC per MySQL

Un Data Source JDBC per connettersi a un database MySQL richiede i seguenti campi:

- Database name
- Username
- Password
- Server
- Database port
- Query (se il campo contiene un'istruzione SQL, deve essere valida)
- Table name

La configurazione dei campi varia a seconda del tipo di Data Source.

The screenshot shows the 'Data Source' configuration window for 'Example JDBC MySQL Glide.sys\_user'. The form is divided into two main sections. The left section contains fields for 'Name' (Example JDBC MySQL Glide.sys\_user), 'Import set table label', 'Import set table name', 'Type' (JDBC), 'Use MID Server', 'Format' (MySQL), 'Database name' (glide), 'Database port', and 'Use Batch Import' (unchecked). The right section contains fields for 'Application' (Global), 'Username' (root), 'Password' (masked with dots), 'Server' (localhost), 'Query' (All Rows from Table), 'Query timeout', 'Connection timeout', 'Table name' (sys\_user), and 'Use last run datetime' (unchecked).

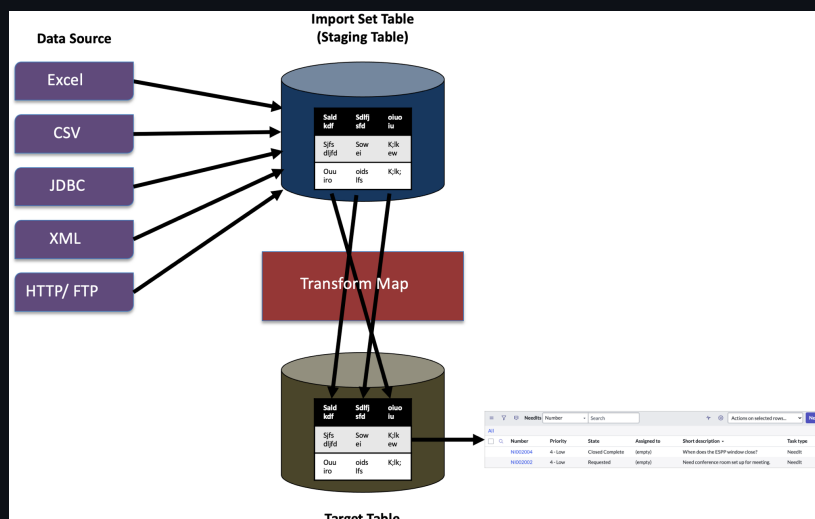
## 7 Data Import Process

I dati non vengono importati direttamente da una Data Source alla tabella target. Il processo prevede diversi passaggi:

1. Caricamento dei dati in una *staging table*
2. Creazione di una *Transform Map*
3. Esecuzione della *Transform* per spostare i dati dalla staging table alla tabella target
4. Verifica dell'integrità dei dati

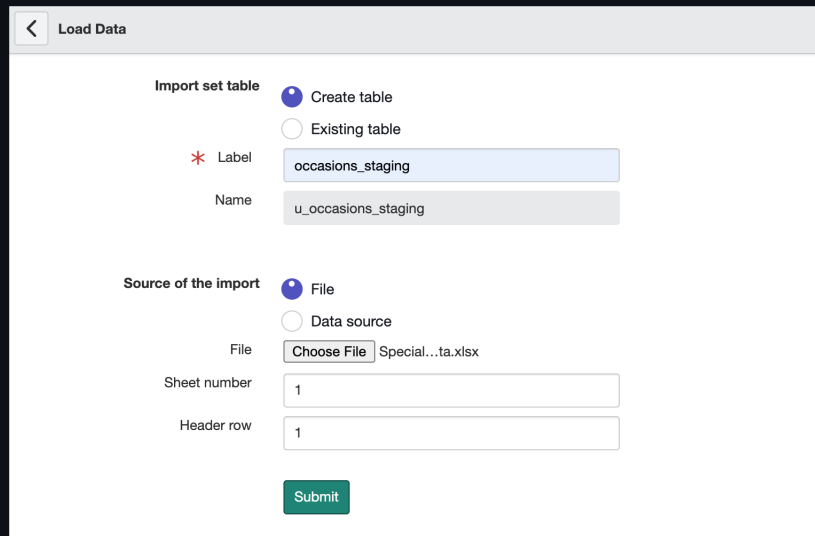
### 7.1 Utilizzo di Studio

Studio viene utilizzato per creare Data Sources e Transform Maps. Tutte le altre operazioni di importazione dei dati vengono eseguite nella finestra principale del browser ServiceNow e non fanno parte di una scoped application.

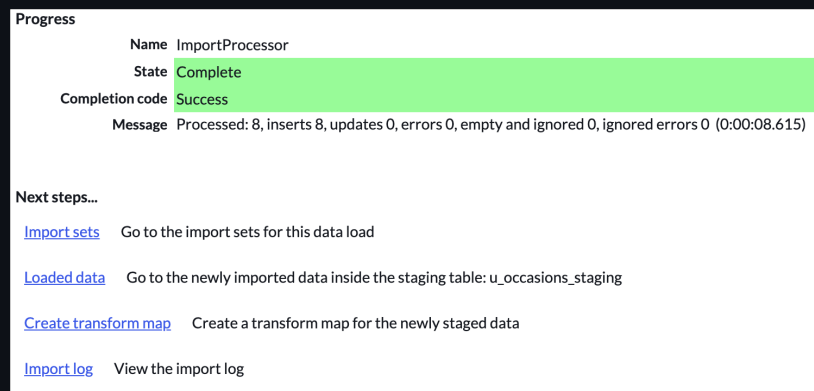


## 8 Load Data

Per caricare dati da una Data Source in una staging table, seguire questi passaggi:



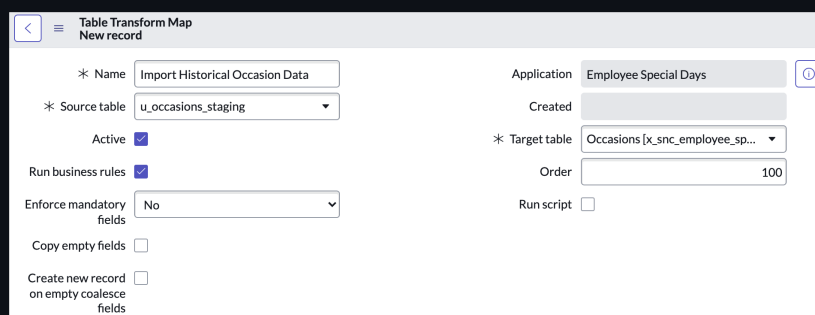
1. Aprire *Load Data* tramite *All menu > System Import Sets > Load Data*
2. Selezionare la Data Source (le opzioni di configurazione variano in base al tipo di Data Source)
3. La staging table viene creata dinamicamente (nell'esempio: `u_occasions_staging`)
4. Cliccare *Submit* per caricare i dati nella staging table
5. Dopo l'importazione, appare una pagina di *Progress* che mostra lo stato dei record importati



6. Cliccare sul link *Create transform map* nella sezione *Next steps...* per mappare i dati dalla staging table alla tabella target

Il caricamento dei dati nella staging table prepara i record per la trasformazione e l'importazione nella tabella finale.

## 9 Create a Transform Map



Una *Transform Map* collega le colonne della staging table ai campi della tabella target. Ogni import richiede almeno una Transform Map.

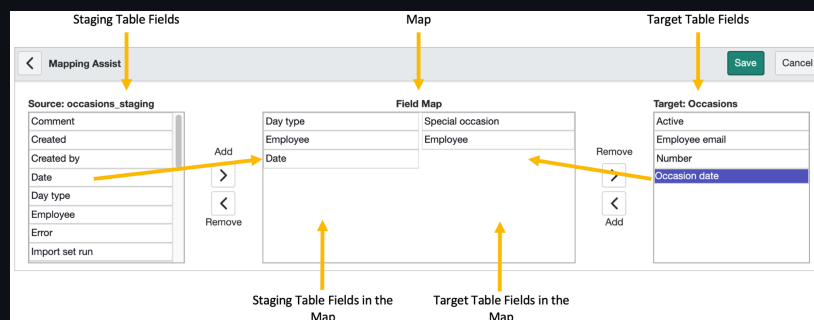
## 9.1 Configurazione del Transform Map

Durante la creazione di una Transform Map, configurare i seguenti parametri:

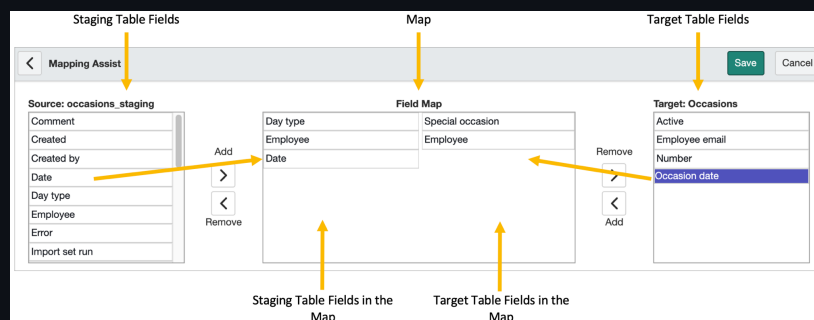
- **Name:** etichetta chiara per identificare la mappa
- **Source table:** tabella di importazione contenente i dati grezzi
- **Active:** abilita la mappa per l'uso
- **Run business rules:** esegue Business Rules, workflow, motori di approvazione, auditing e normalizzazione dei campi durante l'inserimento o aggiornamento dei dati
- **Enforce mandatory fields:** applica i campi obbligatori della tabella target
- **Copy empty fields:** cancella i valori esistenti se il campo in arrivo è vuoto
- **Create new record on empty coalesce fields:** crea un nuovo record se tutti i campi di coalescenza sono vuoti
- **Application:** applicazione a cui appartiene la Transform Map
- **Created:** data di creazione (automatica)
- **Target table:** tabella di destinazione dei dati trasformati
- **Order:** ordine di esecuzione in caso di più mappe applicabili
- **Run script / Script:** logica di trasformazione avanzata dei dati

## 9.2 Mappatura dei Campi

- *Auto Map Matching Fields:* tenta di collegare automaticamente le colonne della staging table a quelle della tabella target



- Per mappature manuali, usare *Mapping Assist*



- Aggiungere o rimuovere campi usando i pulsanti *Add / Remove*
- Spostare campi tramite drag & drop

## 9.3 Verifica dei Dati Mappati

Choose whether to view all fields or only mapped fields

Data Viewer

Show ☐ All Fields ☒ Mapped Fields

occasions\_staging ◀ Viewing: 1 ▶

Field	Value
Employee	Mabel Gerkowski
Day type	birthday
Date	

Staging Table Record

Occurrences ◀ Viewing: 1 ▶

Field	Value
Employee	Alicia Mountjoy
Special occasion	Birthday

Target Table Record

- Utilizzare la sezione *Data Viewer* per confrontare un record della staging table con un record della tabella target
- Controllare valori, formattazione e completezza
- Non tutti i campi devono essere mappati; alcuni possono essere ignorati

Cliccare *Save* per salvare la Transform Map completa.

## 10 Running a Transform Map

Eseguire una *Transform Map* significa importare dati dalla staging table alla tabella target.

### 10.1 Passaggi principali

1. Aprire la Transform Map cliccando sul *Transform Related Link* nel record della Table Transform Map

< Table Transform Map  
Import Historical Occasion Data

\* Name Import Historical Occasion Data

\* Source table u\_occasions\_staging

Active ☒

Run business rules ☒

Enforce mandatory fields No

Copy empty fields ☐

Create new record on empty coalesce fields ☐

Application Employee Special Days ⓘ

Created 12-02 20:57:52

\* Target table Occasions [x\_snc\_employee...]

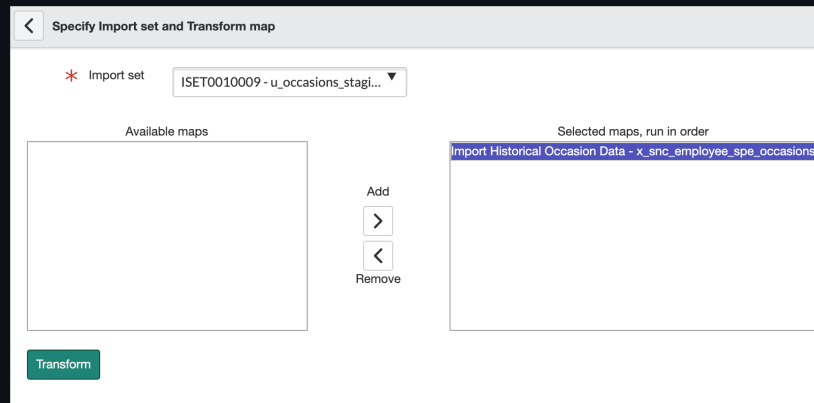
Order 100

Run script ☐

Update Copy Delete

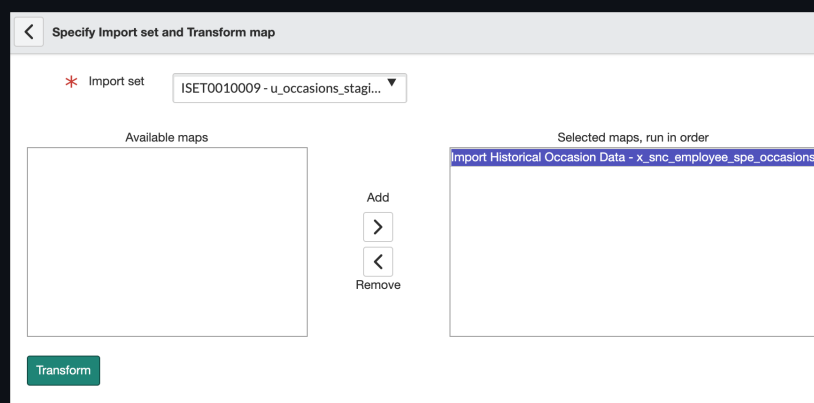
Related Links  
[Auto Map Matching Fields](#)  
[Mapping Assist](#)  
[Transform](#)  
[Index Coalesce Fields](#)

2. Specificare la Transform da eseguire nel modulo *Specify Import Set and Transform Map*



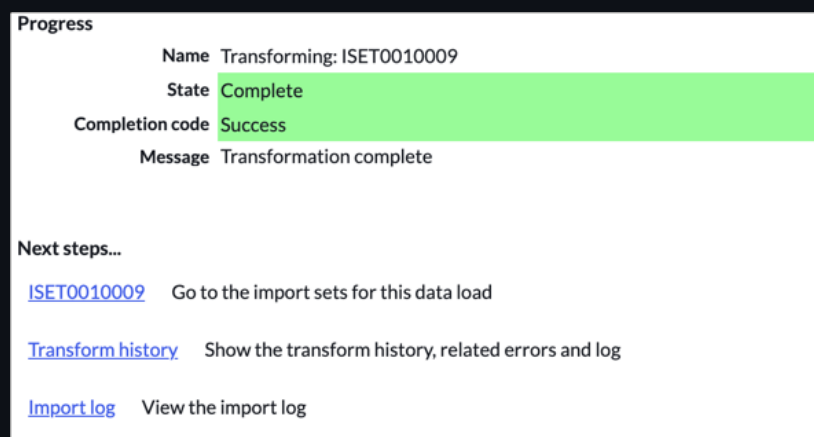
- Verificare che la Transform Map sia presente nella sezione *Selected maps, run in order*
- Cliccare il pulsante *Transform* per avviare l'esecuzione

3. Monitorare il progresso tramite la pagina *Progress*, che mostra lo *State* e il *Completion code*



- Attenzione: il *Completion code* Success indica solo che la trasformazione è stata eseguita correttamente, non che i record siano stati importati senza errori

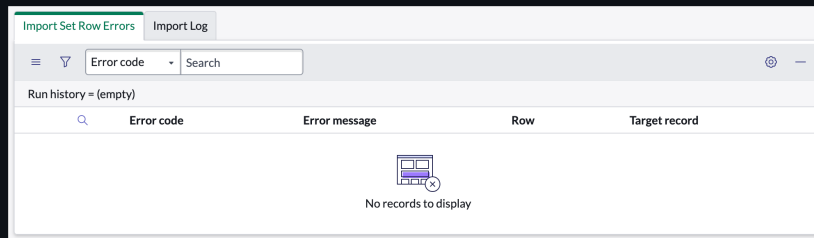
4. Verificare eventuali errori cliccando su *Transform history* e selezionando il timestamp



- Controllare la sezione *Import Set Row Errors* per eventuali errori record per record

## 10.2 Nota Importante

Una transform può completarsi con successo a livello tecnico, ma alcuni record potrebbero non essere importati correttamente. È fondamentale controllare la storia della transform per garantire l'integrità dei dati.

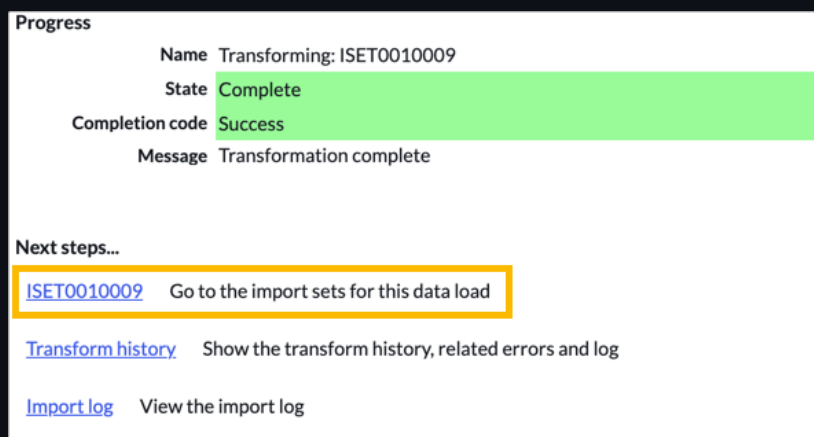


## 11 Verify the Data Integrity

L'ultimo passo nell'importazione dei dati consiste nel verificare che i record contengano i dati previsti nel formato corretto.

### 11.1 Aprire l'Import Set

Cliccare sul link dell'Import Set (es. ISET001XXXX) nella sezione *Next steps...*

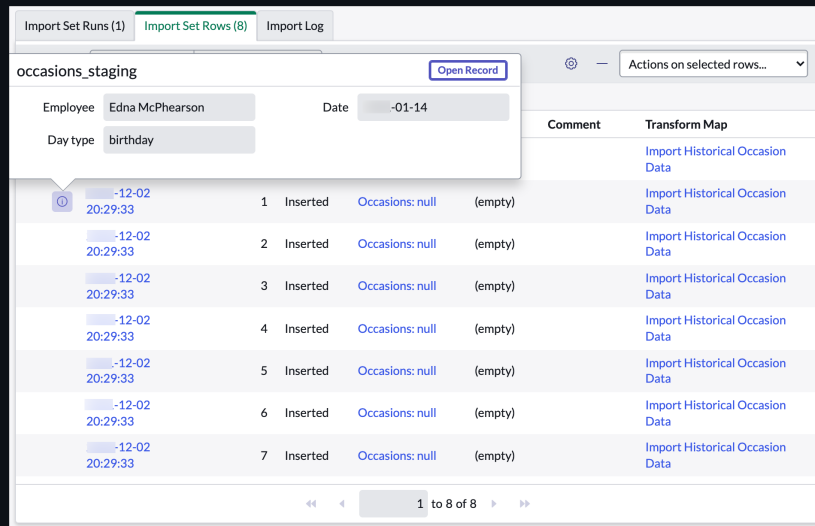


### 11.2 Controllare Import Set Runs

Import Set Runs (1)											
Set = ISET0010009											
	Started	State	Completed	Run time	Total	Inserts	Updates	Processed	Ignored	Skipped	
	2021-12-02 21:05:57	Complete	2021-12-02 21:05:58	1 Second	8	8	0	0	0	0	

- Nella tab *Import Set Runs*, verificare che il numero di record inseriti corrisponda a quello previsto
- Se alcuni record sono stati ignorati o saltati, indagare per comprendere il motivo

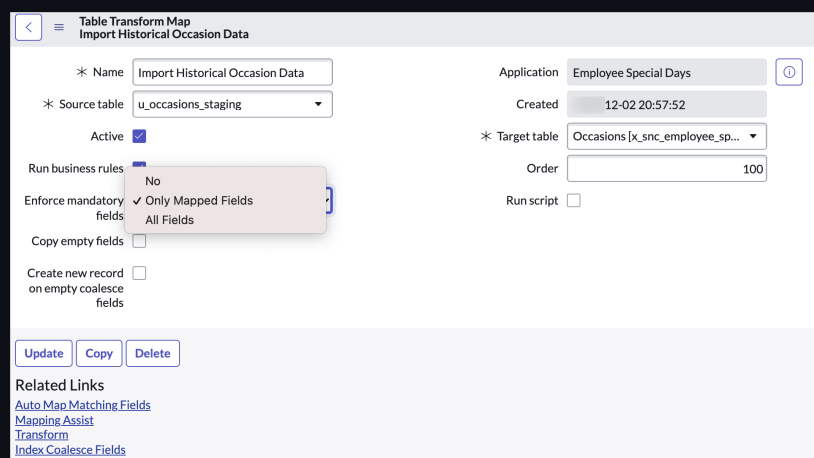
### 11.3 Controllare Import Set Rows



- Passare alla tab *Import Set Rows*
- Passare il mouse su un record e cliccare l'icona *Preview* per vedere i dati importati
- La *Preview* mostra solo i dati importati; cliccare *Open Record* per visualizzare l'intero record

## 11.4 Gestione dei Campi Obbligatori

- Se un campo obbligatorio (es. `Occasion date`) non è valorizzato, l'utente deve inserire un valore prima di salvare il record



- Per imporre valori nei campi obbligatori durante l'importazione, impostare *Enforce mandatory fields* su *All Fields* o *Only Mapped Fields* nella configurazione della Transform Map

## 12 Importing Data from NeedItImportData.csv

In questo modulo si importano i dati dal file `NeedItImportData.csv` nella tabella *NeedIt*. Sono importati quattro campi principali: *Requested for*, *Request type*, *What needed* e *Short Description*.

### 12.1 Preparazione del File CSV

- Aprire il file in un editor o spreadsheet
- Controllare colonne, numero di record e formato dei dati
- Correggere eventuali dati duplicati o incompleti



## 12.2 Caricamento nella Staging Table

---

- Aprire *System Import Sets > Load Data*
- Creare la Import Set Table, ad esempio *Historic NeedIt Data*
- Selezionare il file CSV, impostare Sheet number e Header row
- Cliccare *Submit* per caricare i dati

Progress	
Name	ImportProcessor
State	Complete
Completion code	Success
Message	Processed: 5, inserts 5, updates 0, errors 0, empty and ignored 0, ignored errors 0 (0:00:01.082)

- La pagina *Progress* mostra se l'import è stato completato con successo

## 12.3 Creazione e Configurazione della Transform Map

---

- Nome: *Import Historic NeedIt Data*
- Source table: *Historic NeedIt Data*
- Target table: *NeedIt*
- Opzioni:
  - *Run business rules*: inizialmente selezionata
  - *Enforce mandatory fields*: impostare a No se alcuni campi obbligatori non sono presenti
- Usare *Auto Map Matching Fields* per mappare automaticamente i campi

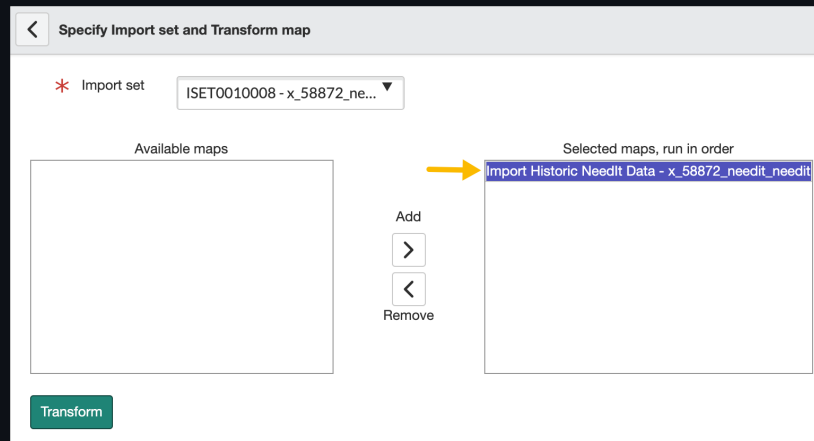
Field Maps (4)		Transform Scripts
		Actions on selected rows... <span>New</span>
<input type="checkbox"/>	Source field	Target field
	u_short_description	short_description
	u_requested_for	u_requested_for
	u_what_needed	u_what_needed
	u_request_type	u_request_type
		Coalesce
		false
		false
		false
		false
		1 to 4 of 4

- Controllare la sezione *Field Maps* per assicurarsi che i quattro campi principali siano mappati correttamente

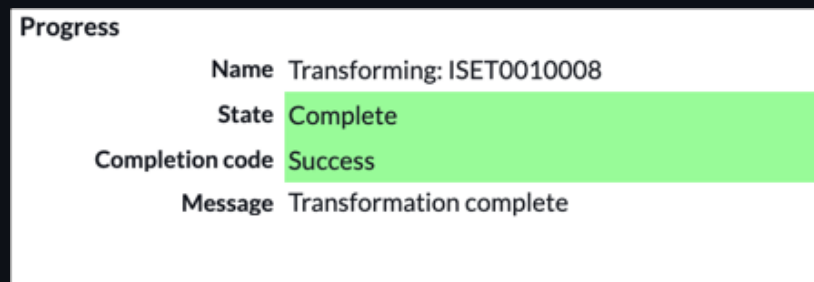
## 12.4 Esecuzione della Transform Map

---

- Aprire *Specify Import Set and Transform Map* e verificare che la map sia nel *Selected maps, run in order*



- Cliccare *Transform* per avviare l'esecuzione



- La pagina *Progress* mostra lo stato della trasformazione

## 12.5 Gestione degli Errori

Transform Histories

Started

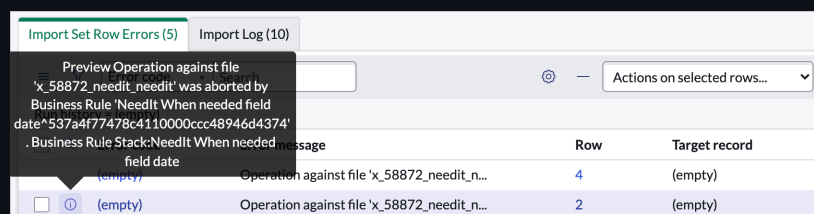
Search

Actions on selected rows...

All > Set Number = ISET0010005

<input type="checkbox"/>	Started	State	Completed	Run time	Set	Import set table	Total	Inserts
<input type="checkbox"/>	-12-02 18:04:59	Complete with errors	-12-02 18:04:59	0 Seconds	ISET0010005	Historic NeedIt Data [x_58872_needit_historic_needit_data]	5	0

- Cliccare su *Transform history* e controllare *Import Set Row Errors*

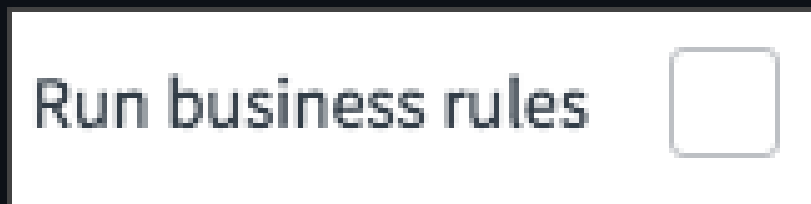


- Gli errori possono derivare da Business Rules o campi obbligatori non mappati
- Nell'esempio, l'errore era causato dalla Business Rule sul campo *When needed*
- Soluzione: disattivare temporaneamente *Run business rules* nella Transform Map

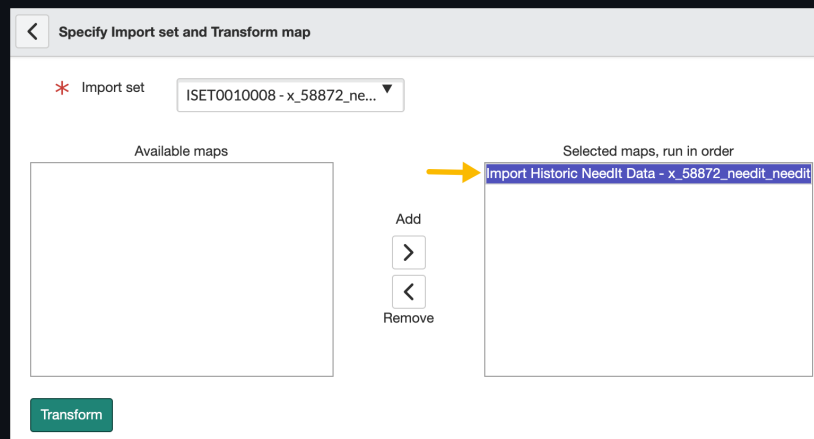
## 12.6 Ripetizione dell'Import e Verifica Finale

- Ricaricare i dati nella staging table

- Eseguire nuovamente la Transform Map con *Run business rules* disattivato



- Verificare *Transform history* e la sezione *Import Set Row Errors* per confermare che non ci siano errori



- Aprire *NeedIt > All* per controllare i record importati

## 12.7 Punti Chiave

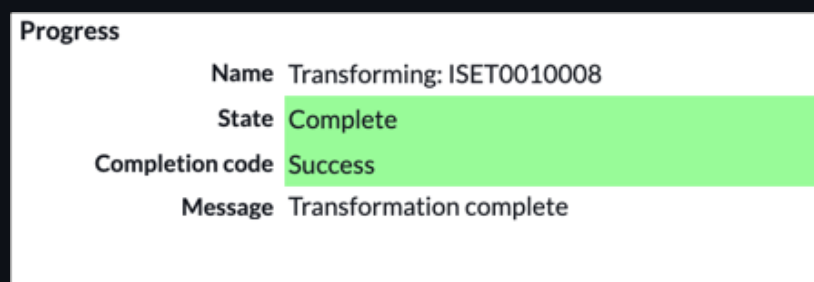
---

- Esaminare sempre i dati prima dell'import
- Errori di import possono derivare da Business Rules o campi obbligatori non mappati
- È possibile modificare la Transform Map senza cambiare il CSV
- Controllare sempre *Import Set Row Errors* per identificare problemi specifici
- Seguire la sequenza corretta: Load Data → Create/Update Transform Map → Run Transform → Verify Data

## 12.8 Verifica Finale dei Record Importati

---

- In *Next steps...*, cliccare sul link *Transform history*
- Lo stato della Transform History dovrebbe essere *Complete*



- La trasformazione non dovrebbe avere errori
- Cliccare sul timestamp per vedere eventuali errori dettagliati

NI002011	4 - Low	Requested	(empty)	Imported record 1	NeedIt
NI002012	4 - Low	Requested	(empty)	Imported record 2	NeedIt
NI002013	4 - Low	Requested	(empty)	Imported record 3	NeedIt
NI002014	4 - Low	Requested	(empty)	Imported record 4	NeedIt
NI002015	4 - Low	Requested	(empty)	Imported record 5	NeedIt

- Controllare la sezione *Import Set Row Errors* (tab); non dovrebbero esserci errori
- Dal menu *All*, aprire *NeedIt > All* per visualizzare i record importati
- I numeri dei record potrebbero differire rispetto agli screenshot d'esempio, ma i dati dovrebbero corrispondere

## 12.9 Importazione di un Campo Data

I campi di tipo *Date* possono generare errori durante l'import se il formato della data nella sorgente non corrisponde a quello atteso dalla tabella target.

Esempio di mismatch di formato:

- CSV: 3/20/68
- ServiceNow: 1968-03-20

**Procedura di correzione:**

1. Aprire la *Transform Map* utilizzata per l'import

Source field	Target field	Coalesce
u_day_type	u_occasion	false
u_employee	u_employee	true
u_date	u_occasion_date	false

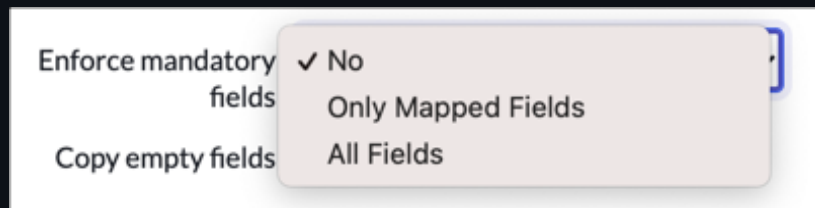
2. Scorrere fino alla lista *Field Maps* e cliccare sul link del campo data nella colonna *Source field*
3. Nel campo *Date format*, specificare il formato della data presente nella staging table

Source field	Target field	Coalesce
u_day_type	u_occasion	false
u_employee	u_employee	true
u_date	u_occasion_date	false

4. Cliccare *Update* per salvare la modifica
5. Eseguire nuovamente la Transform Map per importare correttamente i dati

## 12.10 Gestione dei Campi Obbligatori

ServiceNow non obbliga automaticamente gli import set a fornire valori per tutti i campi obbligatori della tabella target. L'opzione *Enforce mandatory fields* sul Transform Map determina come trattare i campi obbligatori durante l'import.



- **No:** i campi obbligatori della tabella target non richiedono valori
- **Only Mapped Fields:** solo i campi obbligatori mappati dalla staging table devono avere valori
- **All Fields:** tutti i campi obbligatori della tabella target devono avere valori

### Soluzioni per campi obbligatori senza valore

1. Modificare il file sorgente aggiungendo colonne per tutti i campi obbligatori (laborioso se i record sono molti)
2. Scrivere uno script lato server nel Transform Map per popolare i campi target

```
1 (function transformRow(source, target, map, log, isUpdate) {  
2  
3     // If the source data does not have a employee_email field,  
4     // create the target email address from the Employee field value. Email addresses  
5     // have the format firstname.lastname@example.com. Convert the u_employee  
6     // value to lowercase. Replace the space with a . and concatenate with  
7     // @example.com  
8     if(!("employee_email" in source)){  
9         var name = source.u_employee.toLowerCase();  
10        target.employee_email = name.replace(" ", ".") + "@example.com";  
11    }  
12  
13 }(source, target, map, log, action==="update");  
14  
15  
16
```

- L'oggetto **source** è automaticamente istanziato con le proprietà della staging table
- L'oggetto **target** è automaticamente istanziato con le proprietà della tabella target
- Esempio: generare Employee email da Employee con formato `firstname.lastname@example.com`

### DEVELOPER TIP:

- I Field Maps hanno precedenza sugli script del Transform Map
- Non usare la stringa NULL nei dati o negli script; usare Null o null invece

## 12.11 Coalescing dei Record

Prima di importare dati, pianificare come gestire eventuali conflitti tra i dati della sorgente e la tabella target:

- Mantenere il record già presente nella tabella target
- Sovrascrivere il record esistente con i dati della sorgente
- Creare duplicati

L'opzione *Coalesce* in un *Field Map* del Transform Map permette di determinare se una riga della staging table corrisponde a un record nella tabella target.

Source field	Target field	Coalesce
u_day_type	u_occasion	false
u_employee	u_employee	true
u_date	u_occasion_date	false

- Impostare **Coalesce** = **true** per usare un campo come chiave univoca
- Se si usano più campi coalesce, tutti devono corrispondere per identificare un record già esistente
- Se viene trovato un match, il record target viene aggiornato
- Se non viene trovato nessun match, viene creato un nuovo record
- Se non ci sono campi coalesce, tutti i record vengono inseriti automaticamente

**Esempio:** Coalescing solo sul campo `u_employee` non è sufficiente per identificare univocamente i record, perché un dipendente può avere più Occasion (es. Compleanno e Anniversario). Usare più campi coalesce per garantire l'univocità.

## 12.12 Transform Event Scripts

I *Transform Event Scripts* vengono eseguiti durante il processo di trasformazione di un *import set table* verso una *target table*. Questi script permettono di modificare il comportamento della trasformazione in diversi momenti del processo.

Per creare un Transform Event Script, accedere alla lista *Transform Scripts* di un *Transform Map* e cliccare il pulsante **New**.

**Trigger dello Script:** Il campo *When* nello script specifica quando lo script deve essere eseguito nel processo di trasformazione. Le possibili scelte sono:

- **onStart:** eseguito all'inizio dell'import, prima che qualsiasi riga sia letta
- **onBefore:** eseguito all'inizio della trasformazione di una riga, prima che la riga sia trasformata nella target table
- **onAfter:** eseguito alla fine della trasformazione di una riga, dopo che la riga della sorgente è stata trasformata e salvata nella target table
- **onChoiceCreate:** eseguito all'inizio della creazione di un valore di scelta prima che il nuovo valore venga creato
- **onComplete:** eseguito alla fine dell'import, dopo che tutte le righe sono state lette e trasformate
- **onForeignInsert:** eseguito all'inizio della creazione di un record correlato prima che il record venga creato
- **onReject:** eseguito durante la creazione di record esterni o di valori di scelta se il record esterno o il valore di scelta viene rifiutato; l'intera riga della trasformazione non viene salvata

**Esempio pratico:** se si vuole aggiornare solo i record esistenti e non inserire nuovi record durante un coalescing, è necessario usare un *onBefore Transform Script*. Lo script *onBefore* viene eseguito dopo che ServiceNow ha determinato se esiste un record corrispondente nella target table e prima che avvenga l'inserimento.

**Variabili automatiche:**

- **action:** stringa con valori possibili **insert** o **update**, impostata dopo che il coalescing ha determinato se c'è un match
- **ignore:** booleano che, se impostato a **true**, ferma il processo di trasformazione per la riga della sorgente

Per un elenco completo delle variabili disponibili nei *Transform Scripts*, consultare la documentazione ufficiale: [https://docs.servicenow.com/bundle/paris-servicenow-platform/page/administer/import-sets/task/t\\_Map-with-.html](https://docs.servicenow.com/bundle/paris-servicenow-platform/page/administer/import-sets/task/t_Map-with-.html).

## 13 Scheduled Script Executions and Events

In questo modulo verranno affrontati i concetti fondamentali relativi alle *Scheduled Script Executions* (Scheduled Jobs) e al sistema di *Events* di ServiceNow. L'obiettivo è comprendere come automatizzare processi, monitorare le esecuzioni e reagire agli eventi generati dalla piattaforma.

### 13.1 Obiettivi del Modulo

---

Al termine di questo modulo sarai in grado di:

- Creare e testare **Scheduled Script Executions** (Scheduled Jobs)
- Verificare la **schedule** dei job ed accertarti che siano effettivamente pianificati
- Determinare quali **eventi** sono stati generati dal sistema
- Registrare nuovi eventi nel **Event Registry**
- Generare ed effettuare il debug degli eventi
- Rispondere agli eventi tramite **Script Actions**

### 13.2 Scheduled Script Executions (Scheduled Jobs)

---

Le Scheduled Script Executions permettono di automatizzare task ricorrenti tramite script server-side eseguiti secondo una pianificazione definita.

Punti chiave:

- Si trovano in *System Definition > Scheduled Jobs*
- Possono essere creati nuovi job o modificati quelli esistenti
- La pianificazione può essere configurata usando espressioni cron o intervalli predefiniti
- Gli script possono eseguire operazioni come aggiornamenti di record, invio di notifiche o chiamate API

### 13.3 Eventi in ServiceNow

---

Gli eventi sono notifiche generate dalla piattaforma per indicare che un'azione o una condizione si è verificata. Gli script o le notifiche possono reagire agli eventi tramite *Script Actions* o *Email Notifications*.

### 13.3.1 Concetti fondamentali

- Gli eventi vengono registrati tramite la tabella `sysevent`
- Ogni evento ha un nome e un payload (parametri opzionali)
- Gli eventi devono prima essere definiti nell'*Event Registry* prima di poter essere generati

## 13.4 Event Registry

---

L'Event Registry è il repository dove vengono dichiarati gli eventi disponibili.

- Gli eventi vengono registrati alla voce *System Policy > Events > Registry*
- Il nome dell'evento deve essere unico
- È possibile aggiungere una descrizione per documentarne l'uso

## 13.5 Generazione e Debug degli Eventi

---

Gli eventi possono essere generati:

- tramite script server-side usando `gs.eventQueue()`
- tramite business rule
- tramite script actions

Per effettuare il debug:

- abilitare le *Event Logs*
- controllare la tabella `sysevent`
- usare *System Logs > Events*

## 13.6 Script Actions

---

Le *Script Actions* permettono di eseguire automaticamente codice quando viene generato un evento.

- Sono script server-side
- Ascoltano eventi specifici
- Possono modificare record, inviare notifiche o richiamare altre funzioni

## 14 About This Learning Module

**Nota importante:** il contenuto di questo modulo formativo è stato aggiornato l'ultima volta per la release **San Diego**. Potresti quindi trovare differenze rispetto all'interfaccia o alle funzionalità presenti nelle release più recenti come *Utah*.

### 14.1 Applicazioni Utilizzate nel Modulo

---

L'applicazione **Employee Special Days** viene utilizzata come esempio per introdurre concetti chiave nello sviluppo delle applicazioni. **Non viene costruita** durante gli esercizi: serve solo per dimostrazione.

Durante gli esercizi pratici, svilupperai invece l'applicazione **NeedIt**. Tutti i file necessari sono forniti tramite *Source Control*.



## 14.2 Struttura degli Esercizi

---

Gli esercizi del modulo sono identificati in tre modi:

- Icona dedicata nella barra di navigazione.
- Icona esercizio e la parola *Exercise* nella parte superiore della pagina.
- Il titolo della pagina inizia con *Exercise* oppure *Challenge*.

Questa struttura serve per distinguere chiaramente le parti teoriche dalle parti pratiche che richiedono interventi manuali nella piattaforma.

## 14.3 Obiettivo del Modulo

---

L'obiettivo principale è introdurre e consolidare i concetti fondamentali per lo sviluppo di applicazioni su ServiceNow, tra cui:

- Modellazione dati (tabelle, campi, relazioni)
- Importazione dati e Transform Maps
- Business Rules, Script Includes e logica lato server
- Client-side scripting
- Scheduled Script Executions
- Event Management e Script Actions
- Automazioni e processi applicativi

Tutte le attività pratiche si svolgeranno sull'applicazione **NeedIt**.

## 15 Scheduled Script Executions

### 15.1 What is a Scheduled Script Execution?

---

Scheduled Script Executions, chiamati anche **Scheduled Jobs**, sono script lato server che vengono eseguiti automaticamente in un momento specifico oppure in modo ricorrente.

Usali quando hai bisogno che una parte della logica dell'applicazione venga eseguita sulla base di una pianificazione temporale. Alcuni esempi tipici:

- Eseguire periodicamente query su una tabella del database
- Trovare record con la data *Due* scaduta
- Individuare record con scadenze imminenti
- Cercare tutte le richieste di un determinato utente
- Controllare record che sono rimasti troppo a lungo in uno stato come *Resolved*
- Aggiornare record automaticamente, ad esempio:
  - Impostare lo stato su *Closed Complete* dopo un certo tempo
  - Assegnare i record non assegnati a un utente
  - Eliminare o riassegnare tutti i record creati da un determinato utente

## 15.2 Scheduling Behavior

---

Gli Scheduled Script Executions vengono eseguiti secondo una pianificazione configurata dall'utente.

**Developer Tip:** Anche se non è il loro scopo principale, gli Scheduled Script Executions sono super utili per testare la logica server-side, perché puoi configurarli per essere eseguiti immediatamente.

## 16 Creating a Scheduled Script Execution in Studio

### 16.1 Procedure

---

La procedura per aggiungere file a un'applicazione in Studio è identica per qualsiasi tipo di file:

1. Cliccare su **Create Application File**.
2. Selezionare il tipo di file da creare, in questo caso **Scheduled Script Execution**.
3. Configurare il nuovo file.

### 16.2 Scheduled Script Execution Configuration

---

Ogni Scheduled Script Execution condivide alcune configurazioni comuni:

- **Name:** Nome dello Scheduled Script Execution.
- **Active:** Se selezionato, abilita l'esecuzione.
- **Run:** Definisce la periodicità con cui viene eseguita la logica.
- **Application:** Applicazione a cui appartiene lo Scheduled Script Execution.
- **Time zone:** Fuso orario utilizzato per la pianificazione.

### 16.3 Run Options

---

#### 16.3.1 Daily

Configurazione per eseguire la logica ogni giorno a un orario specificato.

#### 16.3.2 Weekly

Configurazione per eseguire la logica in specifici giorni della settimana.

#### 16.3.3 Monthly

Configurazione per eseguire la logica in determinati giorni del mese.

**Developer Tip:** Per eseguire uno Scheduled Script Execution nell'ultimo giorno del mese, impostare il giorno su **31**. Verrà comunque eseguito nell'ultimo giorno effettivo, anche nei mesi con meno di 31 giorni.

#### 16.3.4 Periodically

Configurazione per eseguire la logica a intervalli di tempo regolari. Esempio: esecuzione ogni ora allo scoccare dell'ora.

#### 16.3.5 Once

Esegue la logica una sola volta alla data e ora specificate.

### 16.3.6 On Demand

Esegue la logica quando un amministratore clicca **Execute Now**. Perfetto per testare script rapidamente.

### 16.3.7 Business Calendar

Esegue la logica all'inizio o alla fine degli *Entry* del Business Calendar selezionato.

## 16.4 Scheduled Script Execution Scripts

---

I **Scheduled Script Execution scripts** vengono eseguiti lato server, quindi utilizzano l'API **server-side** di ServiceNow. A meno che un amministratore non clicchi **Execute Now**, gli Scheduled Script Execution non sono innescati da azioni dell'utente.

Questi script non sono associati a record specifici e non hanno accesso agli oggetti **current** o **previous** utilizzati da molti altri script server-side.

### 16.4.1 Campi di scripting

Ogni Scheduled Script Execution possiede due campi principali per lo scripting:

- **Condition:** opzionale; determina se lo script principale deve essere eseguito.
- **Run this script:** contiene la logica server-side da eseguire quando il job viene attivato e la condizione restituisce **true**.

### 16.4.2 Condition Script

- Il campo Condition permette di separare la logica condizionale dallo script principale, evitando di nidificare codice inutilmente.
- Deve impostare la variabile **answer** a **true** per consentire l'esecuzione dello script principale.
- Variabili dichiarate qui possono essere accessibili nello script principale, secondo le regole dello *JavaScript scope*.
- Esempio pratico: verificare se oggi è un giorno feriale o un weekend usando `GlideDateTime.getDayOfWeekLocalTime()` (Monday = 1, Tuesday = 2, ecc.).

### 16.4.3 Run this Script

- Questo script viene eseguito lato server quando il Scheduled Job viene attivato e la condizione ritorna **true**.
- Non ha accesso diretto a record specifici; per interagire con dati, deve fare query sulle tabelle del database.
- Esempio di utilizzo: cercare record nella tabella *Occasion* che corrispondono al mese e giorno odierno, quindi loggare i numeri dei record nell'*Application Log* (System Logs > System Log > Application Logs).

### 16.4.4 Esempio pratico

```
// Condition script
var rightNow = new GlideDateTime();
var dayOfWeek = rightNow.getDayOfWeekLocalTime();
answer = (dayOfWeek >= 1 && dayOfWeek <= 5); // true solo nei giorni feriali

// Run this script
var occGR = new GlideRecord('u_occasion');
occGR.addQuery('month', rightNow.getMonthUTC());
occGR.addQuery('day', rightNow.getDayOfMonthUTC());
occGR.query();
while(occGR.next()){
    gs.log('Occasion record: ' + occGR.number);
}
```

#### 16.4.5 Developer Tips

- Usa il Condition script per filtrare le condizioni di esecuzione e mantenere lo script principale pulito e leggibile.
- Variabili globali nello script Condition possono essere condivise con lo script principale, evitando duplicazioni.
- È possibile testare rapidamente uno Scheduled Script Execution cliccando **Execute Now**, anche se non è la modalità primaria d'uso.

## 16.5 Testing Scheduled Script Executions

---

Per testare un **Scheduled Script Execution**, un sviluppatore potrebbe attendere la prossima esecuzione pianificata, ma questo può allungare molto il ciclo di sviluppo, soprattutto per script trimestrali o annuali.

#### 16.5.1 Eseguire subito uno script

- Utilizzare il pulsante **Execute Now** presente nell'interfaccia del modulo Scheduled Script Execution.
- Questo pulsante è disponibile per tutti i tipi di Scheduled Script Execution, inclusi quelli On Demand, che altrimenti non si eseguono automaticamente.
- Dopo aver cliccato **Execute Now**, lo script viene messo nella coda dello scheduler per essere eseguito il prima possibile.
- Il thread dello scheduler preleva lo Scheduled Script Execution dalla coda e ne esegue la logica.

#### 16.5.2 Condizioni durante il test

- Se lo Scheduled Script Execution include uno script di **Condition**, la variabile `answer` deve comunque essere impostata a `true` per permettere l'esecuzione dello script principale, anche durante il test tramite **Execute Now**.
- Questo permette di simulare condizioni di esecuzione senza modificare il calendario effettivo dello script.

#### 16.5.3 Developer Tips

- **Execute Now** è fondamentale per velocizzare il ciclo di test di script pianificati a lungo termine.
- Controllare i log dell'applicazione (System Logs > System Log > Application Logs) per verificare il corretto funzionamento dello script.

## 16.6 Viewing Scheduled Script Executions on the Schedule

---

Per visualizzare gli **Scheduled Script Executions** previsti per oggi, utilizzare il menu *All* nella finestra principale di ServiceNow e aprire:

System Scheduler > Scheduled Jobs > Today's Scheduled Jobs

#### 16.6.1 Module Overview

Il modulo **Today's Scheduled Jobs** mostra tutti i job programmati da ora fino alla fine della giornata (23:59:59). Non mostra i job già completati nella giornata.

#### 16.6.2 Informazioni visualizzate per ogni job

- **Name**: nome dello Scheduled Script Execution (Scheduled Job)
- **Next action**: momento in cui il job è programmato per essere eseguito
- **Trigger type**: periodicità di esecuzione (daily, weekly, monthly, ecc.)
- **Job ID**: identificatore univoco del job
- **State**: stato attuale, può essere *Ready*, *Running*, *Error* o *Queued*

### 16.6.3 Note Importanti

- I job nello stato *Ready* sono pronti per essere eseguiti dal scheduler quando arriva il momento indicato in *Next action*.
- Se *Ready* appare in rosso, significa che il job non è ancora stato eseguito e il tempo di esecuzione previsto è già passato.
- Se più job sono programmati per lo stesso momento, alcuni potrebbero non essere eseguiti esattamente all'orario previsto, perché il scheduler esegue i job in base alle risorse disponibili.
- Lo scheduler cerca comunque di eseguire ogni job il più vicino possibile all'orario pianificato.

## 16.7 Exercise: Create a Scheduled Script Execution

---

In questo esercizio, creerai e testerai uno **Scheduled Script Execution** per individuare i *NeedIt Tasks* scaduti.

### 16.7.1 Preparazione dei record

Normalmente, i record di *NeedIt Task* vengono creati da un workflow quando le richieste *NeedIt* vengono approvate. Per questo esercizio, creerai manualmente alcuni record:

1. Record scaduto con **State** = **Open**
  - **Assigned to:** Beth Anglin
  - **Due date:** data passata a scelta
  - **Short description:** Overdue
2. Record futuro con **State** = **Open**
  - **Assigned to:** Beth Anglin
  - **Due date:** data futura a scelta
  - **Short description:** Not overdue
3. Record passato con **State** = **Work in Progress**
  - **Assigned to:** Beth Anglin
  - **Due date:** data passata a scelta
  - **Short description:** Overdue
4. Record passato con **State** = **Closed Complete**
  - **Assigned to:** Beth Anglin
  - **Due date:** data passata a scelta
  - **Short description:** Due date in past but record is closed

### 16.7.2 Creazione dello Scheduled Script Execution in Studio

1. Apri **System Applications > Studio** e seleziona l'applicazione *NeedIt*.
2. Clicca su **Create Application File**, scegli **Scheduled Script Execution** come tipo di file.
3. Configura il nuovo Scheduled Script Execution:
  - **Name:** Find Overdue NeedIt Tasks
  - **Active:** selezionato
  - **Run:** Daily
  - **Time zone:** System Time Zone
  - **Time:** 08:00:00
4. Copia e incolla il seguente script nel campo **Run this script:**

```
// Get today's time and date
var rightNow = new GlideDateTime();

// Query the database for NeedIt Task records with Due date field values older
// than the current time. Only return NeedIt Task records that do not have
// a State field value of Closed Complete.
var overdueNITasks = new GlideRecord('x_58872_needit_needit_task');
overdueNITasks.addQuery('due_date','<=',rightNow);
overdueNITasks.addQuery('state','!=',3);

overdueNITasks.query();
// Write a log message for each overdue NeedIt Task Record
while(overdueNITasks.next()){
    gs.info("Overdue NeedIt Task record = " + overdueNITasks.number);
}
```

5. Clicca **Submit** per salvare lo Scheduled Script Execution.

### 16.7.3 Test dello Scheduled Script Execution

- Clicca il pulsante **Execute Now** per forzare l'esecuzione immediata.
- Controlla i log in **System Logs > System Log > Application Logs** per verificare i messaggi relativi ai record *Overdue NeedIt Task*.
- Se non appaiono messaggi, il job potrebbe non aver terminato l'esecuzione; utilizzare il menu **Additional Actions > Refresh List** per aggiornare.

### 16.7.4 Creazione di un Scheduled Script Execution aggiuntivo

- Creare un nuovo Scheduled Script Execution, **Find NeedIt Tasks Due Soon**, per trovare i record con **Due date** tra ora e le prossime 24 ore.
- Impostare **Run: Daily** alle 08:05 AM nel fuso orario di sistema.
- I record devono avere **State** diverso da **Closed Complete**.
- Testare l'esecuzione come indicato sopra, eventualmente creando o modificando i NeedIt Tasks per rispettare i criteri di ricerca.

### 16.7.5 Developer Tips

- I Scheduled Script Executions non sono legati a record specifici e non hanno accesso agli oggetti **current** o **previous**.
- Separare la logica condizionale in un campo **Condition** per evitare nidificazioni profonde di codice.
- Variabili definite nel **Condition** sono accessibili nel campo **Run this script**.
- Per debug rapido, utilizzare l'opzione **Execute Now** invece di attendere la prossima esecuzione programmata.

## 16.8 Events in ServiceNow

---

Gli **Events** in ServiceNow rappresentano indicazioni che qualcosa di significativo è accaduto. Possono essere generati da:

- Script server-side
- Workflow
- Processi interni di ServiceNow
- Azioni utente, ad esempio:
  - Impersonazione di un utente
  - Login
  - Visualizzazione di un record
  - Modifica di un record

### 16.8.1 Caratteristiche degli Events

- Ogni evento viene registrato in una *event queue*.
- Gli eventi di per sé non eseguono azioni; servono come trigger.
- Per reagire agli eventi, devono essere configurate delle risposte.

### 16.8.2 Tipi di Risposta agli Events

- **Email Notification:** invia notifiche agli utenti o gruppi predefiniti
- **Script Action:** esegue uno script server-side per reagire all'evento

### 16.8.3 Processo Generale per Gestire gli Events

1. **Registrare l'evento:** aggiungere l'evento al *Event Registry* (non necessario se l'evento è già registrato).
2. **Generare l'evento:** tramite script, workflow o azione utente.
3. **Rispondere all'evento:** configurare *Script Actions* o notifiche email per reagire all'evento.

### 16.8.4 Developer Tips

- Gli eventi servono come meccanismo di notifica e trigger asincroni, non come esecuzione diretta.
- Usare l'*Event Registry* per evitare duplicazioni e gestire eventi standardizzati.
- Script Actions possono utilizzare variabili legate all'evento, come `event.parm1`, `event.parm2`, ecc., per fornire contesto alla logica di risposta.

## 16.9 Event Registry

---

ServiceNow può rispondere solo agli eventi registrati nell'**Event Registry**. Registrare un evento permette ai processi di ServiceNow di riconoscerlo e reagire in maniera appropriata.

### 16.9.1 Creazione di un Event Registration in Studio

La procedura per aggiungere file a un'applicazione in Studio è identica per tutti i tipi di file:

1. Cliccare su *Create Application File*.
2. Scegliere il tipo di file *Event Registration*.
3. Configurare il nuovo file.

### 16.9.2 Configurazione dell'Evento

- **Suffix:** parte unica del nome dell'evento. Non usare spazi o caratteri speciali.
- **Event name:** nome generato automaticamente per l'evento.
- **Table:** tabella del database associata all'evento.
- **Fired By:** elenca workflow o script che generano l'evento (utile per il troubleshooting, non per il processamento diretto).
- **Description:** descrizione dello scopo dell'evento; importante per la documentazione futura.
- **Queue:** coda di elaborazione dell'evento. Se vuoto, l'evento viene inviato alla coda predefinita. Code personalizzate possono essere configurate.
- **Caller Access:** definisce l'accesso cross-scope all'evento.

### 16.9.3 Developer Tips

- Registrare sempre gli eventi prima di generarli, altrimenti ServiceNow non potrà rispondere.
- Fornire descrizioni chiare e significative per la manutenzione futura.
- Utilizzare correttamente la coda di eventi per gestire priorità o performance.
- Monitorare il campo *Fired By* per capire quali processi o script generano l'evento.

## 16.10 Events in ServiceNow

---

Gli *Events* in ServiceNow rappresentano notifiche che indicano che qualcosa di significativo è accaduto. Possono essere generati da:

- Script server-side
- Workflow
- Processi di sistema ServiceNow
- Azioni utente come login, visualizzazione o modifica di un record

Gli eventi non producono azioni da soli: devono essere gestiti tramite logica aggiuntiva, come notifiche email o *Script Actions*.

### 16.10.1 Event Registry

Un evento deve essere registrato nell'*Event Registry* per poter essere riconosciuto e gestito da ServiceNow.

**Passaggi per registrare un evento in Studio:**

1. Cliccare su *Create Application File*.
2. Selezionare *Event Registration* come tipo di file.
3. Configurare i campi principali:
  - **Suffix:** parte unica del nome dell'evento (senza spazi o caratteri speciali)
  - **Event name:** nome completo generato automaticamente
  - **Table:** tabella di riferimento per l'evento
  - **Fired By:** elenca workflow o script che generano l'evento (solo per troubleshooting)
  - **Description:** descrizione dell'evento
  - **Queue:** coda di processazione dell'evento (lasciare vuoto per la coda predefinita)
  - **Caller Access:** accesso cross-scope

### 16.10.2 Generating Events

Gli eventi possono essere generati in due modi:

- **Server-side script** tramite i metodi `gs.eventQueue()` o `gs.eventQueueScheduled()`
- **Workflow** tramite l'attività *Create Event*

**Esempio di generazione evento via script:**

```
// Genera un evento per una occasione di un dipendente
gs.eventQueue('x_60157_employee_spe.employeeOccasion',
    current,
    current.number,
    gs.getUserName());
```

- `current` è l'oggetto `GlideRecord` del record di riferimento
- `parm1` e `parm2` sono parametri opzionali registrati nel log per il troubleshooting
- È consigliato usare `gs.getUserName()` o `gs.getUserID()` per tracciare l'utente che ha generato l'evento

**Esempio con workflow:** L'attività *Create Event* genera automaticamente un evento basato su `current`, mentre `parm1` e `parm2` possono essere definiti tramite script. Non è necessario specificare la coda, poiché utilizza la coda predefinita.



### 16.10.3 Developer Tips

- Sempre registrare eventi nell'*Event Registry* prima di generarli.
- Usare `parm1` e `parm2` per fornire informazioni utili nel log degli eventi.
- Se l'evento è generato da script server-side, usare `eventQueueScheduled()` se si vuole impostare una scadenza per l'evento.
- Utilizzare eventi per disaccoppiare logica e azioni, migliorando manutenzione e troubleshooting.

## 16.11 Checking for Events

---

Dopo aver generato eventi, è possibile controllarne la creazione e lo stato nel *Event Log*. Per aprire l'Event Log:

- Nel menu principale di ServiceNow, aprire *System Policy > Events > Event Log*.

### 16.11.1 Informazioni visualizzate nell'Event Log

L'Event Log mostra dettagli su tutti gli eventi generati:

- **Created:** data e ora di creazione dell'evento secondo il fuso locale dell'istanza ServiceNow.
- **Name:** nome dell'evento.
- **Parm1 e Parm2:** valori passati quando l'evento è stato generato.
- **Table:** tabella del database a cui si riferisce l'evento.
- **Processed:** data e ora in cui l'evento è stato processato.
- **Processing duration:** tempo impiegato per processare l'evento in millisecondi.
- **Queue:** nome della coda che ha processato l'evento.

### 16.11.2 Note utili

- I valori di `Parm1` e `Parm2` vengono risolti e visualizzati nell'Event Log, utili per troubleshooting.
- Se la colonna `Queue` è vuota, l'evento è stato processato dalla coda predefinita. Se contiene un valore, l'evento è stato processato da una coda personalizzata.
- Un entry con `[object Object]` nella colonna `Table` indica un problema: probabilmente l'oggetto passato con `gs.eventQueue()` apparteneva alla tabella sbagliata.
- Avere il numero del record in `Parm1` o `Parm2` rende più semplice risalire all'oggetto a cui l'evento si riferisce.

### 16.11.3 Developer Tip

Assicurarsi sempre che l'oggetto `GlideRecord` passato a `gs.eventQueue()` appartenga alla tabella corretta per evitare errori di log e garantire una corretta elaborazione.

## 16.12 Responding to Events

---

In ServiceNow, esistono due modi principali per rispondere agli eventi:

- **Email Notification** – invio di notifiche via email (non trattato in dettaglio in questo modulo).
- **Script Action** – handler server-side che esegue codice JavaScript in risposta agli eventi.

### 16.12.1 Creazione di Script Actions in Studio

La procedura per aggiungere file a un'applicazione in Studio è la stessa per tutti i tipi di file:

1. Cliccare su *Create Application File*.
2. Selezionare il tipo di file *Script Action*.
3. Configurare il nuovo Script Action.

### 16.12.2 Configurazione dello Script Action

- **Name:** Nome dello Script Action.
- **Event name:** Evento a cui lo Script Action risponde.
- **Execution Order:** Ordine di esecuzione tra più Script Actions legati allo stesso evento. Tipicamente numeri a 3 cifre, ordine crescente.
- **Active:** Deve essere selezionato per abilitare lo Script Action in runtime. *Developer Tip:* Script Actions non sono attivi di default, ricordarsi di attivarlo.

### 16.12.3 Campi di scripting

- **Condition Script:** condizione JavaScript per determinare se eseguire lo Script. Lo Script viene eseguito solo se la condizione restituisce `true`.
- **Script:** codice server-side JavaScript eseguito se la condizione è vera.

### 16.12.4 Oggetti utili nello Script Action

- `event` – disponibile solo nello Script, permette di accedere a `parm1` e `parm2` con `event.parm1` e `event.parm2`.
- `current` – disponibile sia nel Condition Script che nello Script. Rappresenta l'oggetto passato tramite `gs.eventQueue()`.

### 16.12.5 Nota pratica

- Quando un evento è generato con un Business Rule come:

```
gs.eventQueue('x_60157_employee_spe.employeeOccasion', previous, current.number, gs.getUserName())
```

lo Script Action non riceverà l'oggetto `previous`, ma solo `current`.

- Questo significa che all'interno di uno Script Action, l'oggetto passato è sempre `current`, anche se la Business Rule originaria passava `previous`.

## 16.13 Exercise: Create and Generate an Event

---

In questo esercizio, registrerai, genererai e risponderai a un evento legato ai *NeedIt Task* scaduti.

### 16.13.1 Preparazione dei record

- Crea diversi *NeedIt Task* records, alcuni con *Due date* passate (scaduti).
- Segui la stessa procedura dell'esercizio *Create a Scheduled Script Execution* per creare record manualmente se necessario.

### 16.13.2 Registrazione di un evento

1. Apri Studio e seleziona l'applicazione *NeedIt*.
2. Clicca *Create Application File* e scegli *Event Registration*.
3. Configura l'evento:
  - **Suffix:** overdueNeedItTask
  - **Table:** NeedIt Task [x\_58872\_needit\_needit\_task]
  - **Fired by:** Scheduled Script Execution - Find Overdue NeedIt Tasks
  - **Description:** Questo evento viene generato per ogni NeedIt Task scaduto
4. Clicca *Submit* per registrare l'evento.

### 16.13.3 Modifica dello Scheduled Script Execution

- Apri lo *Scheduled Script Execution Find Overdue NeedIt Tasks*.
- Modifica il `while` loop aggiungendo:

```
gs.eventQueue('x_58872_needit.overdueNeedItTask',  
              overdueNITasks,  
              overdueNITasks.number,  
              gs.getUserName());
```

- Questo genera un evento per ogni NeedIt Task scaduto.
- Salva le modifiche cliccando *Update*.

### 16.13.4 Verifica degli eventi

- Esegui lo script con *Execute Now*.
- Controlla i log in *System Logs > Application Logs* per i messaggi dei record scaduti.
- Controlla l'*Event Log (System Policy > Events > Event Log)* filtrando con `*needit` nel campo *Name*.
- Verifica che Parm1 corrisponda ai record attesi.

### 16.13.5 Creazione dello Script Action

1. In Studio, clicca *Create Application File* e scegli *Script Action*.
2. Configura lo Script Action:
  - **Name:** Update Overdue NeedIt Task Priority
  - **Event name:** x\_58872\_needit.overdueNeedItTask
  - **Active:** selezionato

3. Inserisci il seguente codice nello Script field:

```
// Imposta la priorità del record a 1 - Critical  
current.priority = 1;  
// Aggiorna il record nel database  
current.update();
```

4. Salva con *Submit*.

### 16.13.6 Test finale

- Esegui di nuovo *Find Overdue NeedIt Tasks* con *Execute Now*.
- Apri *NeedIt > NeedIt Tasks* e verifica che i task scaduti abbiano Priority = 1 - Critical.
- Se non funziona, controlla:
  - Event Log per assicurarti che gli eventi siano generati.
  - Che lo Script Action sia *Active*.
  - Assenza di errori nello Script.

### 16.13.7 Estensione: Task in scadenza prossima

- Registra un nuovo evento *dueSoonNeedItTasks* per i task con scadenza entro 24 ore.
- Modifica lo Scheduled Script Execution *Find NeedIt Tasks Due Soon* per generare questo evento.
- Crea uno Script Action *Update Due Soon NeedIt Task Priority* per impostare la priorità dei task a 2 - High.
- Testa assicurandoti di avere almeno un record con scadenza entro 24 ore.

## 16.14 Using Custom Queues - Advanced Topic

---

Quando un'applicazione genera un alto volume di eventi o eventi che richiedono molto tempo per essere processati, è consigliabile utilizzare una *custom queue*.

### 16.14.1 Creazione di una Custom Queue

1. Nel *Event Registry*, popola il campo *Queue* per l'evento.
  - Usa solo lettere minuscole, underscore (`_`) e nessuno spazio o carattere speciale.
2. Esempio: impostando `Queue = my_queue`, gli eventi generati saranno inseriti in questa coda.
3. Nota: gli eventi in una custom queue rimangono bloccati finché non esiste un processo che monitora la coda.

### 16.14.2 Creazione di un processo per monitorare la coda

1. Apri *System Scheduler* > *Scheduled Jobs* > *Scheduled Jobs*.
2. Trova il job *text index events process* e aprilo per modifica.
3. Modifica:
  - **Schedule Item Name** per il nuovo processo.
  - L'argomento di `GlideEventManager()` con il nome della custom queue esatta.
4. Non cliccare *Update* (altrimenti sovrascrivi il job originale).
5. Usa *Additional Actions* > *Insert and Stay* per creare una copia del job.
6. Il nuovo processo monitorerà immediatamente la coda `my_queue`.

### 16.14.3 Invio di eventi alla custom queue

- Rimuovi il valore nel campo *Queue* dell'evento nella registry.
- Usa il quinto parametro della funzione `gs.eventQueue()` per inviare l'evento alla custom queue:

```
gs.eventQueue(  
  'x_60157_employee_spe.employeeOccasion',  
  todaysOccasions,  
  todaysOccasions.number,  
  todaysOccasions.u_employee.name,  
  'my_queue'  
);
```

- Nota: in questo esempio, il quarto parametro è il nome dell'impiegato invece di `gs.getUserName()` per differenziare i log degli eventi.
- Gli eventi generati oggi sono ora processati dalla coda `my_queue`.

### 16.14.4 Considerazioni importanti

- Usare attenzione: non sovrascrivere mai il job originale *text index events process*.
- I nomi delle code devono corrispondere esattamente tra Event Registry e `GlideEventManager()`.
- Le code personalizzate permettono di distribuire e gestire meglio eventi pesanti o numerosi.

## 17 Scheduled Script Executions and Events Module Recap

### 17.1 Scheduled Script Executions (Scheduled Jobs)

---

- Sono script server-side che vengono eseguiti periodicamente secondo la configurazione:
  - Daily
  - Weekly
  - Monthly
  - Periodically
  - Once
  - On demand
- È possibile usare un *Condition script* per determinare se lo script nel campo *Run this Script* debba essere eseguito:
  - Deve restituire `true` o `false`.
  - Le variabili dichiarate nel Condition script sono note al campo Run this Script se lo scope di JavaScript lo consente.
- Non hanno accesso a `current` perché non sono triggerati da record o interazioni utente.
- Per eseguire uno script l'ultimo giorno del mese, configurare un *Monthly Scheduled Script Execution* sul giorno 31.
- Per vedere gli script in programma: *System Scheduler > Scheduled Jobs > Today's Scheduled Jobs*.

### 17.2 Events

---

- Gli eventi indicano che qualcosa di rilevante è avvenuto in ServiceNow.
- Possono essere generati da:
  - Script server-side
  - Workflow
  - Processi di ServiceNow
  - Azioni dell'utente (login, modifica record, ecc.)
- Gli eventi devono essere registrati nel *Event Registry* per poter essere gestiti dai processi di ServiceNow.
- Per generare eventi con `gs.eventQueue()`, passare quattro parametri principali:
  1. Nome dell'evento
  2. Oggetto GlideRecord (tipicamente `current` o record ottenuto da query)
  3. Valore che si risolve in stringa (`parm1`)
  4. Valore che si risolve in stringa (`parm2`)
- Per visualizzare gli eventi generati: *System Policy > Events > Event Log*.
- Il log mostra i valori risolti di `parm1` e `parm2`, utili per il debugging.

## 17.3 Script Actions

---

- Sono script server-side che si eseguono in risposta agli eventi.
- Hanno accesso a:
  - L'oggetto `event`, contenente `parm1` e `parm2`
  - L'oggetto `current` passato all'evento da `gs.eventQueue()`
- Per eventi ad alto volume o che richiedono molto tempo, si consiglia di creare *custom queues* per distribuire il carico e gestire meglio la loro elaborazione.

## 17.4 Conclusione

---

Questo modulo ha introdotto i concetti chiave di Scheduled Script Executions, Eventi e Script Actions, con enfasi su:

- Pianificazione e test degli script periodici
- Generazione e log degli eventi
- Risposta agli eventi tramite Script Actions
- Gestione avanzata con code personalizzate per scenari complessi

# 18 Protezione delle Applicazioni contro Utenti Non Autorizzati

## 18.1 Obiettivi

---

In questo modulo imparerai a:

- Creare e utilizzare *ruoli* (`roles`) e *gruppi* (`groups`)
- Impersonare utenti per testare le autorizzazioni
- Gestire l'accesso alle applicazioni e ai moduli
- Utilizzare i *Controlli di Accesso* (`Access Controls`) per proteggere i record delle tabelle dagli accessi non autorizzati:
  - Creazione
  - Lettura
  - Modifica
  - Eliminazione
- Debug dei Controlli di Accesso
- Utilizzare metodi dell'API di ServiceNow per gestire la sicurezza tramite script

# 19 Introduzione al Modulo di Sicurezza

## 19.1 Informazioni sul Modulo

---

Attenzione: il contenuto di questo modulo è stato aggiornato l'ultima volta per la versione San Diego di ServiceNow. Potresti notare differenze se stai utilizzando la versione Utah.

Durante questo modulo, l'applicazione *Employee Special Days* viene usata come esempio per introdurre concetti e processi relativi alla creazione e gestione di un'applicazione in ServiceNow. **Nota:** non costruirai l'applicazione *Employee Special Days*, serve solo come riferimento.

Gli esercizi pratici saranno invece svolti sull'applicazione *NeedIt*, che svilupperai passo passo.

## 19.2 Come riconoscere gli esercizi

---

Gli esercizi pratici sono indicati in tre modi diversi:

- Icona *Exercise* nel pannello di navigazione
- Icona e la scritta *Exercise* in cima alla pagina
- La parola *Exercise* o *Challenge* nel titolo della pagina

## 19.3 Informazioni sull'applicazione NeedIt

---

L'applicazione *NeedIt* permette agli utenti di richiedere servizi da diversi dipartimenti. Per iniziare, userai il *source control* per avere a disposizione tutti i file necessari dell'applicazione *NeedIt* per completare il modulo.

## 20 Ruoli (Roles)

### 20.1 Cos'è un Ruolo

---

I ruoli controllano l'accesso a funzionalità e capacità nelle applicazioni e nei moduli di ServiceNow. La sicurezza delle applicazioni inizia proprio dai ruoli: senza ruoli correttamente configurati, gli utenti potrebbero accedere a dati o funzioni non autorizzate.

Il sito di documentazione di ServiceNow elenca tutti i ruoli di sistema di base, ma è possibile creare ruoli personalizzati per applicazioni specifiche.

### 20.2 Creare un Ruolo in Studio

---

Per creare un ruolo:

1. Aprire *Studio*.
2. Cliccare *Create Application File*.
3. Selezionare il tipo di file *Role* e cliccare *Create*.

Configurare il ruolo con i seguenti campi:

- **Suffix:** parte unica del nome, viene aggiunta allo scope per formare il campo *Name*.
- **Name:** nome completo del ruolo, visibile quando si assegnano i ruoli.
- **Assignable by:** ruolo che può assegnare questo ruolo ad utenti o gruppi (facoltativo).
- **Requires Subscription:** selezionare *Yes* se il ruolo richiede una licenza o abbonamento.
- **Application:** l'applicazione a cui il ruolo appartiene.
- **Elevated privilege:** se selezionato, il ruolo può elevare la sicurezza e modificare impostazioni ad alta sicurezza.
- **Description:** descrizione dell'utilizzo previsto del ruolo.

**Esempio di configurazione:**

- Suffix: `occasions_user`
- Name: `x_snc_employee_spe.occasions_user`
- Assignable by: (vuoto)

- Requires subscription: No
- Application: Employee Special Days
- Elevated privilege: non selezionato
- Description: Role for users of the Employee Special Days application

## 20.3 Assegnare un Ruolo a un Utente

---

L'assegnazione dei ruoli agli utenti non è parte dei file di un'applicazione: quando un'applicazione viene installata su un'altra istanza, i ruoli degli utenti non vengono automaticamente aggiornati.

Per assegnare un ruolo:

1. Aprire *User Administration* > *Users* dal menu principale di ServiceNow (non Studio).
2. Selezionare un utente dalla lista.
3. Scorrere fino alla sezione *Roles* e cliccare *Edit*.
4. Nel *slushbucket*, selezionare il ruolo da aggiungere. Se non appare nei primi 100 ruoli, usare la funzione di ricerca.
5. Cliccare *Add* per spostare il ruolo nella lista dei ruoli dell'utente.
6. Cliccare *Save*.

## 20.4 Spiegazione Pratica

---

- I ruoli garantiscono che solo utenti autorizzati possano accedere a certe funzioni o dati. - È buona pratica creare ruoli specifici per ogni applicazione e assegnarli solo agli utenti che ne hanno bisogno. - Ruoli con *Elevated Privilege* devono essere assegnati con cautela perché permettono modifiche ad alta sicurezza.

## 21 Gruppi (Groups)

### 21.1 Cos'è un Gruppo

---

Un gruppo è un insieme di utenti che condividono uno scopo comune. I gruppi possono essere utilizzati per:

- Approvare richieste di modifica (*Change Requests*)
- Risolvere incidenti
- Ricevere notifiche email
- Eseguire attività di lavoro (*Work Orders*)

I gruppi rappresentano un modo più efficiente di assegnare ruoli agli utenti. Invece di aggiungere ruoli uno per uno agli utenti, è possibile assegnare i ruoli al gruppo. Tutti i membri del gruppo ereditano automaticamente i ruoli assegnati al gruppo.

### 21.2 Creare un Gruppo

---

Per creare un gruppo:

1. Aprire *User Administration* > *Groups* dal menu principale di ServiceNow (non Studio).
2. Cliccare *New*.



3. Configurare il gruppo con i seguenti campi:

- **Name:** Nome del gruppo.
- **Manager:** Manager o responsabile del gruppo.
- **Group email:** Lista di distribuzione email o contatto principale del gruppo (facoltativo).
- **Parent:** Se il gruppo ha un gruppo padre, eredita i ruoli del gruppo padre. I membri del gruppo figlio non diventano membri del gruppo padre.
- **Description:** Descrizione dello scopo del gruppo.

**Esempio di configurazione:**

- Name: Employee Special Days Users
- Manager: Fred Luddy
- Group email: (vuoto)
- Parent: (vuoto)
- Description: Employee Special Days Users

## 21.3 Assegnare Ruoli a un Gruppo

---

1. Dopo aver creato il gruppo, scorrere fino alla lista *Roles* e cliccare *Edit*.
2. Aggiungere i ruoli desiderati usando lo slushbucket.
3. Cliccare *Save*.

## 21.4 Aggiungere Membri a un Gruppo

---

1. Aprire la lista *Group Members* nel gruppo appena creato.
2. Cliccare *Edit* e selezionare gli utenti da aggiungere tramite lo slushbucket.
3. Cliccare *Save* dopo aver aggiunto tutti gli utenti.

**Nota importante:** L'amministrazione dei gruppi non fa parte dei file dell'applicazione. Quando un'applicazione viene installata su un'altra istanza, i gruppi non vengono creati o aggiornati automaticamente.

## 21.5 Vantaggi dell'uso dei Gruppi

---

- Semplifica la gestione dei ruoli: invece di assegnare ruoli singolarmente, si assegnano ai gruppi.
- Facilita la manutenzione: quando un nuovo utente entra nel gruppo, eredita automaticamente i ruoli del gruppo.
- Gestione gerarchica tramite gruppi padre/figlio per una struttura di autorizzazioni più flessibile.

## 22 Protezione di Applicazioni e Moduli

### 22.1 Controllo accesso alle applicazioni

---

L'accesso a un'applicazione nel menu *All* è controllato dal **ruolo utente** dell'applicazione. Non è sempre necessario limitare l'accesso all'applicazione stessa, se i ruoli vengono applicati ai singoli moduli.

- Per limitare l'accesso a un'applicazione a utenti con un ruolo specifico, aprire l'applicazione in *Studio*.

- Dal menu *File*, selezionare *Settings*.
- Aggiungere un ruolo nel campo *User role*. Solo gli utenti con quel ruolo potranno accedere all'applicazione.

**Esempio:** il campo *User role* ha il valore `x_snc_employee_spe.occasions_user`.

## 22.2 Accesso al menu dell'applicazione

---

L'accesso ai menu delle applicazioni può essere controllato separatamente:

- Nell'*Application Explorer*, aprire *Navigation > Application Menus > <nome applicazione>*.
- Aggiungere uno o più ruoli al campo *Roles* per limitare la visibilità del menu.

**Suggerimento:** Se un utente senza ruolo deve vedere moduli o menu, sia il modulo sia il menu dell'applicazione non devono avere ruoli impostati.

## 22.3 Controllo accesso ai moduli

---

L'accesso ai moduli nel menu *All* è controllato dai ruoli:

- Aprire *System Definition > Modules*.
- Aprire un modulo e scorrere alla sezione *Visibility*.
- Aggiungere uno o più ruoli nel campo *Roles* per limitare l'accesso.

**Esempio:** il modulo *Occasions* ha il ruolo `x_snc_employee_spe.occasions_user`.

## 22.4 Override application menu roles

---

L'opzione *Override application menu roles* permette di concedere accesso a un modulo anche se l'utente non ha il ruolo richiesto per vedere il menu dell'applicazione.

- Esempio pratico: il menu *Special Occasions* richiede il ruolo `admin`, mentre il modulo *Occasions* richiede `x_snc_employee_spe.occasions_user`. Senza l'opzione di override, gli utenti con il ruolo `occasions_user` non possono vedere né il menu né il modulo.
- Se si seleziona l'opzione *Override application menu roles*, gli utenti con il ruolo `occasions_user` possono vedere sia il menu che il modulo, anche senza avere il ruolo richiesto dal menu.

## 22.5 Suggerimenti pratici

---

- Applicare i ruoli più restrittivi solo dove necessario, preferibilmente a livello di modulo.
- Testare sempre gli accessi impersonando utenti con ruoli differenti per verificare che le autorizzazioni siano corrette.
- Ricorda che la combinazione di ruoli a livello di applicazione e modulo può creare conflitti di visibilità; l'opzione di override è utile per risolverli.

## 23 Protezione di Applicazioni e Moduli

### 23.1 Controllo accesso alle applicazioni

---

L'accesso a un'applicazione nel menu *All* è controllato dal **ruolo utente** dell'applicazione. Non è sempre necessario limitare l'accesso all'applicazione stessa, se i ruoli vengono applicati ai singoli moduli.

- Per limitare l'accesso a un'applicazione a utenti con un ruolo specifico, aprire l'applicazione in *Studio*.
- Dal menu *File*, selezionare *Settings*.
- Aggiungere un ruolo nel campo *User role*. Solo gli utenti con quel ruolo potranno accedere all'applicazione.

**Esempio:** il campo *User role* ha il valore `x_snc_employee_spe.occasions_user`.

### 23.2 Accesso al menu dell'applicazione

---

L'accesso ai menu delle applicazioni può essere controllato separatamente:

- Nell'*Application Explorer*, aprire *Navigation > Application Menus > <nome applicazione>*.
- Aggiungere uno o più ruoli al campo *Roles* per limitare la visibilità del menu.

**Suggerimento:** Se un utente senza ruolo deve vedere moduli o menu, sia il modulo sia il menu dell'applicazione non devono avere ruoli impostati.

### 23.3 Controllo accesso ai moduli

---

L'accesso ai moduli nel menu *All* è controllato dai ruoli:

- Aprire *System Definition > Modules*.
- Aprire un modulo e scorrere alla sezione *Visibility*.
- Aggiungere uno o più ruoli nel campo *Roles* per limitare l'accesso.

**Esempio:** il modulo *Occasions* ha il ruolo `x_snc_employee_spe.occasions_user`.

### 23.4 Override application menu roles

---

L'opzione *Override application menu roles* permette di concedere accesso a un modulo anche se l'utente non ha il ruolo richiesto per vedere il menu dell'applicazione.

- Esempio pratico: il menu *Special Occasions* richiede il ruolo `admin`, mentre il modulo *Occasions* richiede `x_snc_employee_spe.occasions_user`. Senza l'opzione di override, gli utenti con il ruolo `occasions_user` non possono vedere né il menu né il modulo.
- Se si seleziona l'opzione *Override application menu roles*, gli utenti con il ruolo `occasions_user` possono vedere sia il menu che il modulo, anche senza avere il ruolo richiesto dal menu.

## 23.5 Suggerimenti pratici

---

- Applicare i ruoli più restrittivi solo dove necessario, preferibilmente a livello di modulo.
- Testare sempre gli accessi impersonando utenti con ruoli differenti per verificare che le autorizzazioni siano corrette.
- Ricorda che la combinazione di ruoli a livello di applicazione e modulo può creare conflitti di visibilità; l'opzione di override è utile per risolverli.

## 24 Impersonificazione degli Utenti

### 24.1 Cos'è l'impersonificazione

---

L'impersonificazione permette a un utente con ruolo **admin** o **impersonator** di diventare temporaneamente un altro utente autenticato. Non è necessario conoscere la password dell'utente impersonato. Durante l'impersonificazione, l'amministratore:

- Vede e può fare esattamente ciò che l'utente impersonato può fare.
- Può testare accessi a moduli, applicazioni e record.
- Verifica che i ruoli e le autorizzazioni siano correttamente configurati.

### 24.2 Procedura pratica

---

1. Nel browser ServiceNow, cliccare sull'icona del profilo nell'angolo superiore destro.
2. Selezionare il menu *Impersonate User*.
3. Nella finestra di dialogo, inserire il nome dell'utente da impersonare nel campo di ricerca.
4. Selezionare l'utente e cliccare *Impersonate user*. L'interfaccia si ricaricherà con i permessi dell'utente impersonato.
5. Dopo i test, terminare l'impersonificazione cliccando sul nome dell'utente impersonato nel banner e selezionando *End Impersonation*.

### 24.3 Suggerimenti pratici

---

- Utilizzare l'impersonificazione per testare nuovi ruoli e Access Controls senza creare utenti di test.
- Ricordarsi di terminare sempre l'impersonificazione per tornare all'utente amministratore.

## 25 Protezione dei Record di una Tabella

### 25.1 Assegnazione dei ruoli alle tabelle

---

Quando si crea una tabella in un'applicazione con **scope**, è necessario assegnare un ruolo utente. Questo ruolo definisce chi può accedere ai record della tabella. Il campo *User role* si trova nella sezione *Controls* del modulo tabella.

- È possibile selezionare un ruolo esistente oppure crearne uno dinamicamente inserendo un nome.
- Per tutte le tabelle scoped, l'opzione *Create access controls* è selezionata di default e non modificabile.

## 25.2 Controlli di Accesso (Access Controls)

---

I Controlli di Accesso definiscono:

1. **Cosa** viene protetto (tabella o campo specifico).
2. **Operazione** che si vuole proteggere: Create, Read, Update, Delete.
3. **Permessi richiesti** per accedere all'oggetto (ruoli o script condizionali).

## 25.3 Access Controls di default

---

Quando si aggiunge una tabella a un'applicazione scoped, ServiceNow crea automaticamente quattro Access Controls di default per la tabella:

- **Create:** Permette di creare nuovi record.
- **Read:** Permette di leggere i record esistenti.
- **Write:** Permette di aggiornare i record.
- **Delete:** Permette di cancellare i record.

## 25.4 Nota sulla sicurezza

---

- ServiceNow adotta la politica *default deny*: l'accesso è negato a meno che non sia esplicitamente consentito da un Access Control.
- I controlli di default proteggono l'intera tabella, ma non i singoli campi. Per proteggere campi specifici, creare Access Controls aggiuntivi a livello di campo.

## 25.5 Esempio pratico

---

Per la tabella **NeedIt**, solo gli utenti con il ruolo `x_snc_employee_spe.occasions_user` possono creare, leggere, aggiornare o cancellare i record.

## 26 Ordine di Valutazione degli Access Controls

### 26.1 Ereditarietà dei Controlli

---

La tabella **NeedIt** estende la tabella **Task**, ereditando quindi i controlli di accesso della tabella genitore. Se non esistono Access Controls specifici per **NeedIt**, i controlli della tabella **Task** vengono valutati.

### 26.2 Ordine di Valutazione degli ACL

---

Quando si valuta l'accesso a un campo, ServiceNow ricerca l'Access Control più specifico prima di passare a quelli più generici. Esempio per il campo *Short description* della tabella **NeedIt**:

1. Controllo specifico su tabella e campo: `x_58872_needit_needit.short_description`

2. Controllo sul campo della tabella genitore: `task.short_description`
3. Qualsiasi tabella con quel campo: `*.short_description`
4. Tabella specifica con wildcard sul campo: `x_58872_needit_needit.*`
5. Tabella genitore con wildcard sul campo: `task.*`
6. Qualsiasi tabella e qualsiasi campo: `.*`

## 26.3 Access Control Configuration Watcher

---

Quando un Access Control viene creato o modificato, il **Configuration Watcher**:

- Analizza l'impatto del controllo sugli altri Access Controls.
- Mostra l'*execution plan* con colori:
  - Rosso: cancellato o disattivato.
  - Giallo: modificato.
  - Verde: aggiunto o attivato.
- Indica se l'Access Control riguarda righe intere o campi specifici.
- Gestisce mascheramento:
  - Masked: il controllo era efficace prima della modifica.
  - Unmasked: il controllo è diventato efficace con la modifica.
- Il link *Show All* permette di visualizzare la ACL completa.

## 27 –None– vs \* negli Access Controls

### 27.1 Definizioni

---

- **–None–**: concede accesso a tutti i record e a tutti i campi.
- **\***: concede accesso a tutti i campi che non hanno ACL specifiche.

### 27.2 Esempio Pratico

---

- Tabella: **Table** con 5 campi.
- Ruoli: **admin** (Fred), **table\_user** (Beth)

**Scenario 1:** –None– per entrambi + ACL su Field 3 per admin

- Fred: accesso a tutti i campi
- Beth: accesso a tutti i campi tranne Field 3

**Scenario 2:** –None– per entrambi + \* per admin + ACL su Field 3 per table\_user

- Fred: accesso a tutti i campi
- Beth: accesso solo a Field 3

## 27.3 Linee Guida

---

1. ACL principalmente positive: usare solo –None–.
2. ACL principalmente negative: combinare –None– e \*.
3. ACL specifiche sui campi hanno priorità sulle generiche.
4. Definire prima accesso record, poi campi sensibili, infine ACL generiche per negazioni.

## 28 Abilitare il Debug degli Access Control

### 28.1 Debug Security Rules

---

Per abilitare il debug degli Access Control in ServiceNow:

1. All menu -> System Security -> Debugging -> Debug Security Rules
2. Il modulo esegue uno script che mostra tutte le informazioni di debug degli Access Control alla fine di ogni pagina nel content frame
3. Informazioni di debug vengono scritte in:
  - Session Log (nuova finestra)
  - Pagine di contenuto (liste e form)
4. Solo gli utenti admin possono accedere a questo modulo
5. Per testare i permessi di altri utenti, impersonare l'utente dopo aver abilitato il debug

### 28.2 Suggerimenti per il Debug

---

- Usare l'impersonation per simulare l'accesso di altri utenti e verificare le regole.
- Le informazioni di debug aiutano a diagnosticare problemi di Access Control e visibilità dei campi.
- Il debug è fondamentale quando più Access Control interagiscono su tabelle o campi ereditati.

## 29 Debug degli Access Control - Pagine di Contenuto

### 29.1 Informazioni di Debug nelle Pagine

---

Quando *Debug Security Rules* è abilitato, uno script permette di visualizzare tutte le informazioni di debug degli Access Control in fondo ad ogni pagina del content frame.

- La stessa sintassi dei messaggi del *Session Log* viene visualizzata nel content frame, con differenze di codice colore.
- Esempio: Beth Anglin non ha accesso al Field 3 della tabella *Table* nell'applicazione *Generic*.

### 29.2 Colori e Simboli

---

- Colori:
  - Verde: Accesso consentito
  - Rosso: Accesso negato
  - Blu: Regola già valutata, risultato memorizzato in cache

- Grigio: Non valutato, tipicamente perché un'altra regola ha negato l'accesso

- **Simboli:**

- : Accesso consentito
- X : Accesso negato

## 29.3 Debug di Singoli Campi

---

- Per il debug di un singolo campo, aprire il form della tabella con *Debug Security Rules* abilitato.
- Un'icona *Debug* appare accanto a ciascun campo. Passando sopra l'icona si vede il numero di messaggi di Access Control per quel campo.
- Cliccando sull'icona è possibile visualizzare le regole applicate al campo specifico.
- Nota: non è possibile utilizzare questo metodo per campi nascosti dagli Access Control.

## 30 Sicurezza tramite Script

### 30.1 API Client-side

---

L'API client-side *GlideUser* (*g\_user*) offre diversi metodi per controllare i ruoli di un utente:

- `hasRole()`
- `hasRoleExactly()`
- `hasRoleFromList()`
- `hasRoles()`

Questi metodi possono essere usati in qualsiasi script client-side, come Client Scripts e UI Policy. Tuttavia, la sicurezza client-side è facilmente aggirabile, quindi non va usata per proteggere dati sensibili.

### 30.2 API Server-side

---

L'API server-side *GlideSystem* (*gs*) offre metodi per ottenere informazioni sull'utente e controllare i ruoli:

- `getUser()`
- `getUserID()`
- `getUserName()`
- `hasRole()`
- `isLoggedIn()`
- `isInteractive()`
- `getSession()`

L'API *GlideElement* server-side consente di verificare se un utente può accedere a specifici *GlideRecord*:

- `canCreate()`
- `canRead()`
- `canWrite()`

Questi metodi possono essere usati in qualsiasi script server-side, come Business Rules o Script Includes, e garantiscono maggiore sicurezza rispetto ai controlli client-side.



## 30.3 Note Importanti

---

- Gli script di sicurezza non fanno parte del modulo *Debug Security Rules* e devono essere testati separatamente.
- Per la massima sicurezza, è consigliato usare Access Controls per proteggere dati sensibili.

## 31 Client-side Scripting: Obiettivi del Modulo

### 31.1 Cosa Imparerai

---

- Descrivere lo scopo degli script client-side e identificare cosa possono fare sul browser dell'utente.
- Creare e testare i *Client Scripts* per implementare logiche personalizzate nei form.
- Creare e testare script di *UI Policy* per controllare comportamento e aspetto dei campi.
- Utilizzare le API `GlideForm` e `GlideUser` all'interno degli script.
- Determinare quando conviene utilizzare una *UI Policy* rispetto a un *Client Script*.

## 32 Informazioni sul Modulo

### 32.1 Panoramica

---

Il contenuto di questo modulo formativo è stato aggiornato per l'ultima volta nella release *San Diego* di ServiceNow e non riflette completamente le modifiche introdotte con la release *Utah*. Alcune differenze nell'interfaccia o nei comportamenti della piattaforma sono quindi normali.

Esempi presenti nelle sezioni concettuali vengono utilizzati esclusivamente a scopo illustrativo; non è necessario ricrearli. Gli esercizi pratici prevedono invece lo sviluppo dell'applicazione *NeedIt*.

### 32.2 Identificazione degli Esercizi

---

Gli esercizi sono contrassegnati da:

- un'icona specifica nel riquadro di navigazione;
- l'icona e la parola *Exercise* nella parte superiore della pagina;
- la presenza delle parole *Exercise* o *Challenge* nel titolo della pagina.

Le pagine relative agli esercizi evidenziano chiaramente il prefisso *EXERCISE*, le icone e gli elementi di navigazione associati.

### 32.3 Applicazione NeedIt

---

L'applicazione *NeedIt* permette agli utenti di richiedere servizi provenienti da diversi dipartimenti. Per gli esercizi di questo modulo, l'applicazione viene inizializzata tramite *source control*, che fornisce tutti i file necessari alla sua implementazione.

## 33 Introduzione al Client-side Scripting

### 33.1 Panoramica

---

Gli script in ServiceNow possono appartenere a due categorie principali: *client-side* e *server-side*. Questo modulo si concentra sugli script lato client, i quali vengono eseguiti direttamente nel browser dell'utente e sono utilizzati per gestire il comportamento dei form e dei relativi campi.

### 33.2 Funzionalità degli Script Client-side

---

Gli script client-side permettono di controllare e personalizzare l'interazione dell'utente con i form. Alcune funzionalità tipiche includono:

- posizionare automaticamente il cursore in un campo all'apertura del form;
- mostrare avvisi, conferme e messaggi;
- compilare dinamicamente un campo sulla base del valore di un altro;
- evidenziare campi specifici;
- validare i dati immessi dall'utente;
- modificare le opzioni di una choice list;
- nascondere o mostrare campi o intere sezioni.

### 33.3 Tipi di Script affrontati nel Modulo

---

Nel modulo verranno trattati e messi in pratica due principali tipologie di script client-side:

- **Client Scripts**, che consentono un controllo dettagliato del comportamento dei form tramite logica personalizzata;
- **UI Policy Scripts**, utili per definire rapidamente regole di visibilità, obbligatorietà e comportamento dei campi.

## 34 Tipi di Client Script

### 34.1 Panoramica

---

I Client Script vengono eseguiti nel browser dell'utente e possono attivarsi in momenti diversi durante l'interazione con un form. Ogni tipo di Client Script risponde ad un evento specifico del ciclo di vita del form.

### 34.2 onLoad

---

Gli script di tipo **onLoad** vengono eseguiti quando un form viene caricato. Sono utilizzati per modificare l'aspetto o il contenuto del form prima che l'utente possa interagire con esso. È consigliabile limitarne l'uso poiché possono rallentare i tempi di caricamento del form.

## 34.3 onChange

---

Gli script di tipo `onChange` vengono eseguiti quando il valore di un campo cambia. Sono utili per reagire dinamicamente ai valori inseriti dall'utente e per aggiornare altri campi o proprietà del form in base a tali cambiamenti.

## 34.4 onSubmit

---

Gli script di tipo `onSubmit` vengono eseguiti quando il form viene inviato. Sono utilizzati principalmente per validare i dati prima del salvataggio e, se necessario, per impedire l'invio quando le informazioni non risultano corrette o complete.

## 34.5 Nota sugli onCellEdit

---

Il campo **Type** dei Client Script include anche l'opzione `onCellEdit`, dedicata alle modifiche sulle liste. Questo tipo non viene trattato all'interno del presente modulo.

# 35 Creazione dei Client Script

## 35.1 Procedura di Creazione

---

Per aggiungere un Client Script tramite Studio, la procedura è sempre la stessa:

1. Cliccare su *Create Application File*.
2. Selezionare il tipo di file *Client Script*.
3. Configurare tutte le opzioni dello script.

## 35.2 Opzioni di Configurazione

---

Ogni Client Script deve essere configurato per definire quando e dove deve essere eseguito.

- **Name:** Nome dello script, preferibilmente secondo una convenzione chiara.
- **Table:** Tabella a cui lo script si applica.
- **UI Type:** Contesto di esecuzione (Desktop/Tablet, Mobile/Service Portal, oppure All).
- **Type:** Evento che attiva lo script (`onLoad`, `onChange`, `onSubmit`).
- **Field Name:** Campo monitorato, disponibile solo per script di tipo `onChange`.
- **Active:** Indica se lo script è abilitato.
- **Inherited:** Se selezionato, lo script si applica anche alle tabelle che ereditano dalla tabella indicata.
- **Global:** Se selezionato, lo script si applica a tutte le viste.
- **View:** Vista specifica a cui lo script si applica; visibile solo quando l'opzione *Global* non è attivata.

## 35.3 Nota per gli Sviluppatori

---

Quando si modificano script lato client, è consigliato ricaricare la finestra principale del browser per assicurarsi che venga caricata la versione più recente della logica.

## 36 Client Script Field

Quando si crea un Client Script in ServiceNow, il campo **Script** viene popolato automaticamente con un template in base al tipo di script selezionato. Questo template serve come punto di partenza per aggiungere la logica desiderata.

### 36.1 onLoad Client Script

---

Il template generato è:

```
function onLoad() {  
    // Inserire qui la logica dello script  
}
```

Esegue la logica client-side **al caricamento del form**, prima che l'utente possa interagire con esso. Esempio di uso: mostrare un alert di benvenuto o messaggi informativi.

### 36.2 onSubmit Client Script

---

Il template generato è:

```
function onSubmit() {  
    // Inserire qui la logica dello script  
}
```

Esegue la logica client-side **quando l'utente invia il form**. Può prevenire l'invio del form se alcune condizioni non sono soddisfatte. Esempio: alert di conferma prima dell'invio del record.

### 36.3 onChange Client Script

---

Il template generato è:

```
function onChange(control, oldValue, newValue, isLoading, isTemplate) {  
    // Inserire qui la logica dello script  
}
```

Parametri passati automaticamente da ServiceNow:

- **control**: campo configurato per lo script
- **oldValue**: valore del campo al caricamento del form
- **newValue**: valore attuale modificato dall'utente
- **isLoading**: true se il cambiamento avviene durante il caricamento del form
- **isTemplate**: true se il cambiamento è dovuto a un template

### 36.3.1 Best Practices

- Documentare il codice con commenti.
- Evitare che lo script `onChange` si esegua durante il caricamento del form o quando il valore è vuoto:

```
if(isLoading || newValue === '' || isTemplate) {  
    return;  
}
```

- Ricordare che `oldValue` resta fisso fino al prossimo reload del form.

## 37 Esercizio: Test dei Client Script Types

In questo esercizio si testano i Client Script di tipo **onLoad**, **onChange** e **onSubmit** utilizzando script predefiniti dell'applicazione NeedIt.

### 37.1 Preparazione

---

1. Aprire l'applicazione NeedIt in Studio.
2. Rendere attivi i seguenti Client Script:
  - NeedIt onLoad Example
  - NeedIt onChange Example
  - NeedIt onSubmit Example
3. Controllare la tabella a cui si applicano e il campo *Field name* per lo script onChange.

### 37.2 Test onLoad Client Script

---

- Aprire un record NeedIt.
- L>alert deve comparire: *Thank you for loading the NeedIt form....*
- Finché l>alert è aperto, il form non è modificabile.
- Chiudere l>alert per interagire con il form.

### 37.3 Test onChange Client Script

---

- Modificare un campo qualsiasi tranne *Short description*: nessun alert.
- Modificare il campo *Short description* e cliccare fuori dal campo: comparirà un alert con *oldValue* e *newValue*.
- Nota: *oldValue* non cambia fino al reload del form.

### 37.4 Test onSubmit Client Script

---

- Cliccare il pulsante Update sul form.
- Comparirà un alert di conferma prima dell'invio dei dati.

## 37.5 Pulizia

---

- Tornare in Studio e rendere inattivi i Client Script testati.

## 38 La Classe GlideForm (g\_form)

L'API client-side `GlideForm` fornisce metodi per gestire form e campi nei Client Script. Tutti i metodi sono accessibili tramite l'oggetto globale `g_form`.

### 38.1 Funzioni principali

---

- `getValue('campo')`: ottiene il valore del campo sul form (non dal database)
- `setVisible('campo', boolean)`: nasconde o mostra un campo
- `setReadOnly('campo', boolean)`: rende un campo in sola lettura
- `addInfoMessage('messaggio')`: mostra un messaggio informativo all'utente
- `addErrorMessage('messaggio')`: mostra un messaggio di errore
- `showFieldMsg('campo', 'messaggio', 'tipo')`: mostra un messaggio accanto a un campo
- `clearMessages()`: cancella tutti i messaggi
- `addOption('campo', 'valore', 'label')`, `clearOptions('campo')`: gestiscono le liste a scelta
- `getSections()`, `getSectionName(id)`: informazioni sulle sezioni del form

### 38.2 Label vs Name

---

- **Label**: nome leggibile sul form, visibile all'utente
- **Name**: nome tecnico usato negli script, sempre minuscolo e senza spazi

### 38.3 Esempio

---

```
alert(g_form.getValue('short_description'));
```

Mostra il valore del campo "Short description" sul form.

## 39 La Classe GlideUser (g\_user)

L'API client-side `GlideUser` permette di ottenere informazioni sull'utente corrente e sui suoi ruoli. Tutti i metodi e proprietà sono accessibili tramite l'oggetto globale `g_user`.

### 39.1 Proprietà principali

---

- `firstName`: nome dell'utente
- `lastName`: cognome dell'utente
- `userName`: nome utente
- `userID`: `sys_id` del record utente (32 caratteri unici)

## 39.2 Metodi principali

---

- `getFullName()`: restituisce il nome completo (`firstName + lastName`)
- `hasRole('role_name')`: verifica se l'utente ha il ruolo assegnato (include ruoli impliciti come `admin`)
- `hasRoleExactly('role_name')`: verifica solo i ruoli esplicitamente assegnati

## 39.3 Esempio

---

```
alert("g_user.firstName = " + g_user.firstName
      + ", \n g_user.lastName = " + g_user.lastName
      + ", \n g_user.userName = " + g_user.userName
      + ", \n g_user.userID = " + g_user.userID);
```

Visualizza informazioni sull'utente attualmente connesso.

## 40 Esercizio: Creazione di Client Scripts

In questo esercizio si creano e testano due Client Scripts per l'applicazione *NeedIt*.

### 40.1 Client Script: NeedIt Request Type Options (onChange)

---

- Aggiorna le opzioni del campo **What needed** in base al valore del campo **Request type**.
- Pulisce la lista di opzioni: `g_form.clearOptions('u_what_needed')`
- Aggiunge opzioni rilevanti e l'opzione "Other": `g_form.addOption()`
- Mantiene il valore pre-caricato se il form è in modalità caricamento e non è un nuovo record: `g_form.setValue()`

### 40.2 Client Script: NeedIt Set Requested For (onLoad)

---

- Imposta automaticamente il campo **Requested for** all'utente attualmente loggato per nuovi record.
- Controlla se il record è nuovo: `g_form.isNewRecord()`
- Imposta il valore con `g_form.setValue('u_requested_for', g_user.userID)`

### 40.3 Flusso di Test

---

1. Creare i Client Scripts in Studio tramite "Create Application File".
2. Configurare Tabella, Tipo (onChange/onLoad), UI Type e Field name (per onChange).
3. Aprire/modificare record NeedIt:
  - Cambiare **Request type** e verificare che le scelte di **What needed** si aggiornino dinamicamente.
  - Creare un nuovo record e verificare che **Requested for** sia impostato all'utente loggato.

## 41 UI Policies

Le UI Policies sono logiche lato client simili ai Client Scripts, ma possono essere configurate senza codice. Permettono di controllare il comportamento dei campi su un form, come obbligatorietà, visibilità o sola lettura.

### 41.1 Creazione e Configurazione

---

- Creazione: Studio → Create Application File → UI Policy
- View: Default (campi essenziali), Advanced (tutti i campi e trigger)
- Campi principali:
  - **Table**: tabella target
  - **Application**: scope dell'applicazione
  - **Active**: attiva/disattiva la policy
  - **Short description**: descrizione della policy (identificazione)
  - **Order**: ordine di valutazione tra più UI Policies
  - **Condition**: condizione che attiva la policy
  - **Global/View**: applicabilità a tutte le view o a una specifica
  - **On load**: valuta la condizione anche al caricamento del form
  - **Reverse if false**: applica azione opposta se condizione falsa
  - **Inherit**: estende la policy alle tabelle derivate
- Note:
  - Se Condition non ha valore, la policy si attiva sempre al cambiamento di campo
  - Order: consigliati valori distanziati (100, 200, 300) per facilitare modifiche future

### 41.2 UI Policy Actions

---

Le UI Policy Actions definiscono come una UI Policy modifica campi o liste correlate sul form lato client, senza necessità di scripting.

- **Campi**: possono impostare tre attributi
  - Mandatory: campo obbligatorio
  - Visible: campo visibile/nascosto
  - Read only: campo sola lettura
- **Configurazione**:
  - Selezionare il campo
  - Impostare l'attributo su True, False, Leave alone
  - Opzione **Clear the field value** per azzerare valori esistenti
- **Reverse if false**:
  - Se selezionata: applica azione opposta quando la condizione è falsa
  - Se non selezionata: nessuna azione applicata
- **Liste correlate**: usare la sezione *UI Policy Related List Actions*
  - Selezionare la lista correlata
  - Impostare visibilità: True, False, Leave alone
  - Esempio: nascondere lista *Incidents* nel form *Problem* se la condizione è vera



## 41.3 UI Policy Scripts

---

I UI Policy Scripts estendono le funzionalità delle UI Policy Actions, permettendo logiche condizionali personalizzate.

- **Attivazione:** selezionare *Run scripts* nella *Advanced View*.
- **Campi di script:**
  - Execute if true: eseguito quando la condizione è vera
  - Execute if false: eseguito quando la condizione è falsa (richiede *Reverse if false* selezionato)
- **Template:** logica scritta dentro la funzione `onCondition()`.
- **Piattaforme:** per default Desktop/Tablet; modificabile con *Run scripts in UI type* (Desktop, Mobile/Service Portal, All)
- **Scopo:** eseguire azioni diverse dalla gestione dei campi (mandatory, read-only, visibile) o condizioni più complesse

## 41.4 Exercise: Create UI Policies

---

In questo esercizio si crea una UI Policy per il campo *Other* nel modulo NeedIt.

### 1. Aggiungere il campo:

- Tipo: String
- Label: Other
- Name: `u_other`
- Posizionarlo sotto il campo *What needed*

### 2. Creare la UI Policy:

- Table: NeedIt
- Active: true
- Short Description: NeedIt show or hide Other field
- Condition: What needed is Other
- Global: true
- Reverse if false: true
- On load: true

### 3. UI Policy Action:

- Field: Other
- Mandatory: true
- Visible: true

### 4. UI Policy Script:

- Execute if true:

```
function onCondition() {
    g_form.showFieldMsg('u_other','Briefly explain what you need.','info');
}
```
- Execute if false:

```
function onCondition() {
    g_form.hideFieldMsg('u_other');
}
```

### 5. Test:

- Impostare *What needed* su Other: il campo appare, diventa obbligatorio e mostra il messaggio.
- Impostare *What needed* su un altro valore: il campo scompare e il messaggio viene rimosso.

