STC EasyBP® Abap4Web

Sommario

[Abstract 3](#_Toc39580568)

[EasyBP® Abap4Web 3](#_Toc39580569)

[EasyBP® Abap4Web - Customizing 4](#_Toc39580570)

[Architettura EasyBP® Abap4Web 4](#_Toc39580571)

[EasyBP® Abap4Web - Definizione delle principali tabelle 5](#_Toc39580572)

[Tabella dei Moduli 🡪 MODUL 5](#_Toc39580573)

[Tabella dei Model 🡪 MODEL 5](#_Toc39580574)

[Tipi modello 🡪 MTYPE 5](#_Toc39580575)

[Azioni Server 🡪 ACTN 5](#_Toc39580576)

[Azioni input/Output 🡪 ACTIO 5](#_Toc39580577)

[Associazione Azioni Model 🡪 ACTMD 5](#_Toc39580578)

[Tabella Componente generico 🡪 CTYPE 6](#_Toc39580579)

[Tabella Default Feature 🡪 CTFEA 6](#_Toc39580580)

[Tabella configurazione 🡪 CTPAN 6](#_Toc39580581)

[Profilo di Navigazione 🡪 NAVPR 6](#_Toc39580582)

[Gruppi di navigazione 🡪 NAVGR e NAVGM 6](#_Toc39580583)

[Testi profilo Navigazione 🡪 NAVGT 6](#_Toc39580584)

[Gerarchia profilo di navigazione 🡪 NAVHR 6](#_Toc39580585)

[Tabella delle applicazioni 🡪 APPS 6](#_Toc39580586)

[Applicazione – Moduli 🡪 APMOD 6](#_Toc39580587)

[Modulo – Action 🡪 MOACT 7](#_Toc39580588)

[Modulo - Model 🡪 MODMO 7](#_Toc39580589)

[Components 🡪 COMPS 8](#_Toc39580590)

[Tabella dei Testi 🡪 COMPT 8](#_Toc39580591)

[Tabella configurazione 🡪 PANS 8](#_Toc39580592)

[Tabella degli stati 🡪 CMPMD e CMPFU 8](#_Toc39580593)

[Gerarchia Componenti 🡪 CMPHR 8](#_Toc39580594)

[Tabella Data element 🡪 DTEL 9](#_Toc39580595)

[EasyBP® Abap4Web - Cluster table and Transactions 10](#_Toc39580596)

[Application & Navigation - Zstc\_Easybp\_Appl 10](#_Toc39580597)

[Modules Configuration - Zstc\_Easybp\_Modulcnf 10](#_Toc39580598)

[Models Configurations - Zstc\_Easybp\_ModConf 10](#_Toc39580599)

[Server Actions Configurations - Zstc\_Easybp\_ServAct 11](#_Toc39580600)

[Components Configuration - Zstc\_Easybp\_Comp 11](#_Toc39580601)

[Component Types - Zstc\_Easybp\_Ctype 11](#_Toc39580602)

[Component-Specific Configurations Zstc\_Easybp\_Cspconf 12](#_Toc39580603)

[Model Types Zstc\_Easybp\_Mtype 12](#_Toc39580604)

[STC EasyBP® Abap4Web - WorkBench 13](#_Toc39580605)

[Strutturazione del dato 13](#_Toc39580606)

[Comportamento Applicazione 13](#_Toc39580607)

[Enhancement implementation 🡪 Applicazione 13](#_Toc39580608)

[BaDI Implementation 🡪 Moduli 13](#_Toc39580609)

[BaDI Implementation 🡪 Model 13](#_Toc39580610)

[Programma generatore applicazioni Base 14](#_Toc39580611)

[Strutturazione del dato 14](#_Toc39580612)

[Workbench 16](#_Toc39580613)

# Abstract

Abap4Web è un interfaccia web modulare costruita per consentire lo sviluppo di applicazioni web basate su SAP in modo facile e veloce. Per costruire le applicazioni non è necessario avere competenze di linguaggi per il web, ma esclusivamente SAP base difatti lo sviluppo viene fatto interamente in SAP, eccezione fatta solo per alcune configurazioni delle quali si vedrà in seguito.

## EasyBP® Abap4Web

L’interfaccia Abap4Web sviluppata interamente in abap, consta di due livelli:

* il front-end che si occupa della costruzione degli oggetti grafici e dell’interazione con l’utente
* il server ovvero il sistema SAP che invia al front-end la configurazione degli oggetti grafici e si occupa di rispondere alle richieste del front-end.

Sia il front-end che il server, scompongono il flusso in sotto elementi per consentire una gestione modulare sia della rappresentazione del flusso stesso che delle interazioni utente-sistema, analizziamo gli oggetti principali:

* Applicazioni (APPS) – consiste nell’esecuzione base di ABAP4Web, al suo interno possono essere rappresentate entità SAP operazioni su di esse, caratteristiche associate, dipendenze, etc in base alle esigenze del cliente.
* Model (MODEL) – definiscono l’entità SAP, o una caratteristica associata, che si vuole rappresentare viene definito da una struttura Dictionary per la rappresentazione del dato e generalmente da una BAdI per definirne il comportamento.
* Moduli (MODUL) – oggetto base dell’applicazione di ABAP4Web, il singolo mattone che rappresenta una macro operazione sul flusso (esempio ricerca di un BP, visualizzazione/creazione/modifica di un ordine), ad un modulo possono essere associati più model e più componenti, sono componibili ed indipendenti dall’applicazione, ad essi è generalmente associata una BAdI per definirne il comportamento rispetto alle richieste del Front-End.
* Azioni (ACTN) – costituiscono le interazioni tra front-end e server associate al modulo, ogni Azione può contenere uno o più model o semplicemente l’attuale record selezionato
* Componenti (COMPS) – costituiscono tutti gli oggetti grafici che vengono visualizzati nell’interfaccia
	+ widget quali form, griglie, o ricerca che rappresentano il dato ovvero il model ad essi associato,
	+ bottoni per indicare una possibile azione da lanciare sul modulo o sul widget.

I componenti possono essere collegati gerarchicamente tra di loro per costruire l’aspetto grafico desiderato, possono avere anche degli stati di visualizzazione (SHOW, EDIT, CREATE, DEFAULT)

* Componenti Generici (CTYPE)- sono i componenti generici, ovvero i tipi dei componenti indicati sopra
* Feature (CTFEA) – non sono legate all’applicazione ma al componente generico. Generalmente sono automaticamente assegnate per tipo di componente (es bottone “aggiungi filtro” sul componente di ricerca).

L’applicazione può essere rappresentata come un contenitore con dentro altri contenitori che sarebbero i moduli oggetti indipendenti dall’app, che possono interagire tra di loro e contenere al proprio interno hanno gli oggetti che rappresentano il dato e quelli che interagiscono su di esso

Figura 1 Schema applicazione Abap4Web

# EasyBP® Abap4Web - Customizing

## Architettura EasyBP® Abap4Web

La struttura fisica sul database di Abap4Web può essere rappresentata dal seguente schema.



Figura 2 Schema Abap4Web

Nello schema si è voluto evidenziare le tabelle riferite agli oggetti citati nel paragrafo precedente.

## EasyBP® Abap4Web - Definizione delle principali tabelle

### Tabella dei Moduli 🡪 MODUL

La tabella **MODUL** definisce il modulo, ovvero l’oggetto base di EasyBP, questo elemento si occupa di gestire una o più operazioni su di un’entità o raccolta di entità, se si vuole sfruttare la modularità della piattaforma meglio assegnare una operazione per modulo, in modo da poterlo riutilizzare in più app.

Nella tabella è richiesto la definizione dell’ID del modulo. Nella tabella è possibile specificare anche se il modulo viene utilizzato come search-help in tal caso va indicato il model per la ricerca e il campo risultato come da se11 per il search-help classici.

### Tabella dei Model 🡪 MODEL

In questa tabella si definisce il modello che tratterà i dati del database, la definizione del modello oltre al definire un codice identificativo consta anche dell’attribuzione di un tipo che oltre a caratterizzarlo come oggetto, definisce anche la struttura del dato struttura che può essere anche fisica sul dictionary ed indicata sempre in tabella. Il tipo del model è definito nella tabella \***MTYPE**.

Per caratterizzare il model in maniera più avanzata è possibile definire le relazioni nella tabella **MREL** e **MDREL** dove i model vengono messi in relazione tra di loro è può essere specificata una chiave per la lettura dei figli.

### Tipi modello 🡪 MTYPE

è la tabella che descrive i tipi possibili di model, in particolar modo abbiamo i tipi Collection, LMF, Model di Ricerca, Model Struttura, Model Tabella, Root Model (Struttura), Table Model – DDIC, Variant Manager, ad ogni tipo è associato una classe di gestione del model.

Inoltre al model definito può essere associato un altro model o una chiave di ricerca utile soprattutto nel caso di model di tipo tabella. Particolarità della tabella model è che può essere anche definita una classe ad-hoc per personalizzare ancor di più l’oggetto, in questo caso sarebbe consono estendere la classe del tipo base indicata in **MTYPE**.

### Azioni Server 🡪 ACTN

Le azioni corrispondono agli eventi che causano il cambiamento di stato del model o il cambio di modulo, le azioni vengono definite nella tabella **ACTN** vengono associate ad una coppia Modulo – Task per indicare appunto il posizionamento dell’azione, a quale modulo fa riferimento e il task per la gestione dei dati. Tali task i possono definire come delle modalità di esecuzione del module, esempio in visualizzazione in modifica etc. La coppia deve essere definita anche nella tabella **CMPMD**, task possibili sono Show, Edit, Creation, Default, si analizzerà in seguito il comportamento.

Nella Tabella **ACTN** è possibile caratterizzare il comportamento dell’azione tramite i parametri presenti in essa, di particolare interesse è il parametro auto-interval che consente all’azione di ripetersi in un intervallo di tempo prestabilito in millisecondi

Per completare la caratterizzazione dell’azione ci sono altre due tabelle la **ACTIO** e la **ACTMD**

#### Azioni input/Output 🡪 ACTIO

La tabella **ACTIO** descrive le direzioni dell’azione ovvero verso chi invia l’evento, chi riceve l’evento un esempio è un’azione di un’applicazione online che invia un evento al server e/o riceve dal server.

#### Associazione Azioni Model 🡪 ACTMD

La tabella **ACTMD** concettualmente è simile alla precedente perché tratta sempre delle direzioni dell’azione ma questa volta si parla di cosa invia e di cosa riceve il modulo allo scatenarsi dell’azione, ovvero l’associazione dell’evento al modulo che invia/riceve.

### Tabella Componente generico 🡪 CTYPE

La tabella in oggetto definisce il singolo componente generico ovvero il tipo componente che può essere la griglia, il form, la ricerca. Nella tabella si associa una classe ABAP per gestire il comportamento e la configurazione del componente, quindi classe CSS, titolo definibile nella **CTYPT**, e testi propri del componente tabella **CTYTX**

#### Tabella Default Feature 🡪 CTFEA

Tabella dipendente dalla CTYPE definisce le feature di default associate ad un componente esempio sono le azioni di esportazione sulle griglie, a questa tabella fanno capo altre due tabelle per descrivere il titolo e i testi della feature (**CTFTX**) e i modi (**CTFMD**) ovvero se la feature deve essere attiva in un particolare stato di visualizzazione

#### Tabella configurazione 🡪 CTPAN

Tabella di configurazione iniziale per il componente locale, va indicato l’id del component e le condizioni

### Profilo di Navigazione 🡪 NAVPR

il profilo di navigazione è paragonabile ad un menù di navigazione, viene definito nella **NAVPR**, rappresenta la parte a destra dell’applicazione mediante la quale è possibile navigare tra le pagine dell’applicazione.

### Gruppi di navigazione 🡪 NAVGR e NAVGM

I gruppi di navigazione sono le cartelle sul menu di navigazione posto a destra della WebUI ad ogni gruppo è possibile associare uno o più moduli, sempre in questa tabella quindi possiamo trovare più volte il Gruppo ma con posizioni diverse, una per modulo associato. Da notare che i moduli che vengono associati in questa tabella compariranno nel profilo di navigazione e saranno navigabili come una pagina dell’applicazione, invece se il modulo associato all’applicazione non è associato ad un gruppo di navigazione sarà visualizzato come popup.

#### Testi profilo Navigazione 🡪 NAVGT

Definisce i titoli associati al gruppo di navigazione, è possibile definire il titolo in più lingue, il testo estratto in base alla lingua di sistema verrà visualizzato a schermo. Da segnalare che la non presenza del testo può essere causa di errore nella visualizzazione del gruppo.

#### Gerarchia profilo di navigazione 🡪 NAVHR

Mette in relazione i gruppi con il profilo o con i gruppi stessi.

### Tabella delle applicazioni 🡪 APPS

La tabella **APPS** definisce il nome tecnico dell’applicazione della WebUI, il suo legame con il relativo profilo di navigazione e alcune configurazioni generiche che si ripercuotono su tutti i moduli componenti associati.

La tabella **APPS** è la prima che viene caricata dal processo che costruisce l’applicazione, da questa si diramano le associazioni ai vari elementi gestite principalmente dalle tabelle descritte nei successivi paragrafi.

#### Applicazione – Moduli 🡪 APMOD

Tabella che lega l’applicazione ai moduli, tale associazione seppur indiretta è fornita dal legame con il profilo di navigazione, ma mentre quest’ultima permette la navigazione lineare tra i moduli, nella **APMOD** devono essere assegnati tutti i moduli associati all’applicazione altrimenti l’esecuzione andrà in errore.

#### Modulo – Action 🡪 MOACT

La tabella **MOACT** lega la coppia Applicazione-Modulo alle server action o azioni, da non confondere con la direzione dell’action che è descritta nella tabella **ACTN**

#### Modulo - Model 🡪 MODMO

Mette in relazione la coppia Applicazione-Modulo con i model. Se in tabella non vengono associati i model richiesti l’applicazione andrà in errore.

### Components 🡪 COMPS

I componenti rappresentano tutti gli oggetti grafici della webui la tabella che li descrive è la **COMPS**. In questa tabella gli oggetti direttamente collegati ad un’applicazione tramite il campo ***APP\_ID***, vengono identificati da un id e associati ad un tipo, definito nella **CTYPE**. Anche qui si ha la possibilità di estendere la classe del tipo principale con una ad-hoc per ulteriori personalizzazioni.

Nella tabella vanno riportati sia i moduli che i button che i widget, ovvero quegli oggetti che graficamente andranno a rappresentare il dato, e come si può intuire possono essere di vario tipo, in base alla struttura del dato (form <-> struttura, griglia <-> tabella …), il tipo è sempre definito nella **CTYPE**

La compilazione della tabella è soggetta al tipo del componente, difatti oltre ai campi già menzionati, si può indicare

* per i Button la relativa action nel campo action\_id
* per i widget ci sarà da indicare il model che si occupa della gestione del dato e il percorso che richiama il model, in caso si tratti di model con più relazioni.
* Per le feature l’id della feature corrispondente.

#### Tabella dei Testi 🡪 COMPT

Nella **COMPT** vano riportati i testi dei vari componenti, è possibile il multi-lingua specificando il codice della lingua.

Da dire che il nome de componente può essere generalizzato nella **CTYCT** associato al tipo generico di componente, se si desidera un nome ad-hoc per il componente invece è necessario definirlo nella **COMPT**.

**Importante**: se il nome non è definito e il componente non è di tipo generico, (es. button) il componente non verrà attivato e non sarà visibile.

#### Tabella configurazione 🡪 PANS

La **CTPAN** indica le condizioni di partenza (collassabile collassato auto-expand) del componente e le dimensioni per il tipo di componente generico, per il componente locale vi è la **CTPANS.**

 volute

#### Tabella degli stati 🡪 CMPMD e CMPFU

La **CMPD** descrive lo stato del componente, che può essere un modulo, un button, un widget… ovvero definisce il modo nel quale il component è visibile, tale coppia è presente anche nell’action per richiamare il modulo nello stato voluto, la coppia presente in **ACTN** deve essere presente in questa tabella come anche eventuali altri stati del modulo.

La **CMPFU** invece si riferisce sempre agli stati di visibilità (Default, Creazione, Modifica e Visualizza) ma questa volta si riferisce ai campi del componente, in più è possibile specificare i campi che sono editabili e/o richiesti

Sempre lato customizing dopo aver definito gli oggetti grafici e il modello dati si procederà con il relazionare tali oggetti mediante le seguenti tabelle.

#### Gerarchia Componenti 🡪 CMPHR

Rispetto alle altre tabelle di relazioni tra oggetti quest’ultima è più complessa in quanto mette in relazioni i vari componenti, ovvero gestisce l’intera gerarchia a partire dal modulo ai widget presenti in esso, specificare una posizione a partire da 1, come anche i widget all’azione associata al widget. Per aspetti di analisi al modulo va associato il componente “\_NOTIFICATION”.

Si può dire che la tabella rappresenti il disegno del modulo, in quanto lega i vari oggetti al suo interno in maniera gerarchica.

### Tabella Data element 🡪 DTEL

Per la webUi è possibile specificare dei data element per creare dei match code o dei menu a tendina, la tabella che li definisce è la **DTEL**, in essa è possibile specificare il data element, possibilmente lo stesso nome del D.E presente nel dictionary, il tipo di dato, un modulo di ricerca nel caso sia stato creato un modulo di tipo match code e l’utilizzo della tabella di ritorno che può essere considerata come un Match Code (application tables) o come menu a tendina (customizing tables), nel campo VALUE\_AS\_HELP. La descrizione del formato ci presenta l'uscita del dato.

La ricerca dei valori può essere realizzata in tre modi:

* con un modulo di tipo match code,
* con la tabella di controllo del data element, collegata nel dictionary al corrispondente data element, in questo caso si dovrà specificare i campi, valore, lingua e testo nei tre corrispondenti campi della tabella
* oppure con una classe che implementa l'interfaccia ZSTC\_WEBUI\_IF\_DROP\_DOWN in tal modo è possibile implementare l'unico metodo dell'interfaccia per valorizzare la tabella in uscita con la coppia codice valore desiderata.

Altri campi presenti in tabella descrivono la visualizzazione del dato, il campo per il quale ordinare il risultato ed il tipo di ordinamento.

### EasyBP® Abap4Web - Cluster table and Transactions

Per modificare le tabelle sono stati realizzati delle view cluster e le corrispondenti transazioni che consentono la modifica di gruppi di tabelle raggruppate in base alle loro dipendenze. Di seguito le transazioni e i gruppi di tabelle in gioco.

#### Application & Navigation - Zstc\_Easybp\_Appl



Figura 3 - Application & Navigation - Zstc\_Easybp\_Appl

La transazione Zstc\_Easybp\_Appl consente dimodificare l’anagrafica dell’applicazione e la sua configurazione generale immagini titoli. e i profili e gruppi di navigazione questi ultimi contengono i moduli e consentono la navigazione all’interno dell’app

#### Modules Configuration - Zstc\_Easybp\_Modulcnf



Figura 4 - Modules Configuration - Zstc\_Easybp\_Modulcnf

La transazione Zstc\_Easybp\_Modulcnf consente di modificare l’anagrafica dei moduli, le app associate e quindi gli elementi dipendenti dalla coppia Applicazione-Modulo ovvero: model, action e sottocomponenti e modi di visualizzazione. Da notare che per copiare un modulo su di un'altra applicazione basterà copiare l’inserimento nella APMOD per ricopiare tutti i sotto elementi

#### Models Configurations - Zstc\_Easybp\_ModConf



Figura 5 - Models Configurations - Zstc\_Easybp\_ModConf

La transazione Zstc\_Easybp\_ModConf consente la definizione dei model e l’associazione ad una struttura dictionary che rappresenta l’informazione da mostrare

#### Server Actions Configurations - Zstc\_Easybp\_ServAct

La transazione Zstc\_Easybp\_ServAct consente la definizione delle azioni quindi la personalizzazione dei testi, la direzione dell’azione e il contenuto ovvero il model o ad esempio nel caso delle griglie il record selezionato (parametro GETALL deselezionato)

Figura 6 - Server Actions Configurations - Zstc\_Easybp\_ServAct

#### Components Configuration - Zstc\_Easybp\_Comp



Figura 7 - Components Configuration - Zstc\_Easybp\_Comp

La transazione Zstc\_Easybp\_Comp consente la definizione dei componenti associati all’app, quindi le loro relazioni sempre in questa transazione è possibile modificare gli stati di visualizzazione e stabilire quali campi rendere editabili o obbligatori in creazione/modifica

#### Component Types - Zstc\_Easybp\_Ctype



Figura 8 - Component Types - Zstc\_Easybp\_Ctype

La transazione in oggetto definisce i componenti generici ovvero i tipi dei componenti e le feature generiche associate al tipo componente (es “aggiungi filtro” sul componente di ricerca, le esportazioni sulle griglie…)

#### Component-Specific Configurations Zstc\_Easybp\_Cspconf

La transazione Zstc\_Easybp\_Cspconf dettaglia i componenti creati in precedenza in particolare è possibile aggiungere configurazioni ai widget quali dimensioni di partenza.

#### Model Types Zstc\_Easybp\_Mtype

La Transazione Zstc\_Easybp\_Mtype si occupa invece del tipo di modulo.

# STC EasyBP® Abap4Web - WorkBench

### Strutturazione del dato

La prima operazione lato workbench è la formattazione del dato ovvero come e cosa rappresentare. Partendo dal progetto di quel che si vuole realizzare si analizzano i requisiti e si struttura il dato, creandone una rappresentazione sul dictionary, che possono consistere in strutture per i singoli record e/o tipi tabella per i record multipli o le collection. Tali strutture andranno mappate nelle tabelle viste nel paragrafo Customizing associate al relativo model /widget.

### Comportamento Applicazione

Il processo che esegue la WebUI, parte dalla tabella APPS e scendendo l’albero rappresentato in figura 1, legge i moduli associati ad essa e per ognuno di loro caricherà i Model, le azioni, i componenti associati e le loro caratteristiche/relazioni.

Il comportamento di questi oggetti verrà implementato tramite l’estensione delle Badi e delle classi associate alla WebUI.

### Enhancement implementation 🡪 Applicazione

Per ogni applicazione sarà creato un enhancement implementation a partire dall’enhancement spot ZSTC\_WebUI. Una volta creato l’enhancement implementation si può proseguire alla creazione delle badi implementation che definiranno il comportamento dei moduli associati all’applicazione.

### BaDI Implementation 🡪 Moduli

Per ogni Modulo sarà creata una BaDI implementation definite dalla ZSTC\_WEBUI\_BADI\_MODULE ed una classe di implementazione avente come super classe la ZSTC\_WEBUI\_BADI\_ROOT\_MODULE e come interfaccia oltre alla IF\_BADI\_INDERFACE la ZSTC\_WEBUI\_IF\_BADI\_MODUL. Quest’ultima porterà in dote i metodi da estendere per implementare il modulo.

Per completare l’associazione della BaDI al modulo è necessario creare un filtro avente come valore il corrispondente identificativo MODULEID definito nella tabella MODUL. Le azioni del modulo sono invece mappate tramite un attributo d’instanza la tabella T\_ACTION. Per inserire in tabella le azioni andrà ridefinito il metodo MAP\_ACTIONS della classe inserendo nella tabella le azioni associate al modulo nella tabella MOACT, ad ogni azione deve essere associato un ‘token’ che, concatenato alla stringa ‘DO\_ACTION\_’ servirà a richiamare il metodo corrispondente all’azione. L’interfaccia \*MODUL porta in dote i metodi base (DO\_ACTION\_SEARCH, DO\_ACTION\_GET\_DETAIL, DO\_ACTION\_CREATE …) ma naturalmente è possibile implementare altri metodi che abbiano la stessa interfaccia.

I metodi richiamati hanno come input l’azione (classe ZSTC\_WEBUI\_FWK\_ACTION) e una tabella per la gestione dei messaggi in exporting (ZSTC\_WEBUI\_T\_WEB\_MESSAGE).

L’azione in genere legge il model (action->get\_Model(model\_id)) in input, lavora i dati in base alle caratteristiche date al model e a restituisce un model in output (action->set\_Model(model\_id) ).

Per l’identificazione del model rifarsi alla tabella ACTMD valore send per l’input, receive per l’output, per le caratteristiche rifarsi alla tabella model e relative.

### BaDI Implementation 🡪 Model

Anche per l’implementazione dei Model si ricorre alle BaDI implementation create sempre nello stesso enhancement, a cambiare sarà la definizione di BaDI, la **ZSTC\_WEBUI\_BADI\_MODEL**, la super-classe **ZSTC\_WEBUI\_BADI\_ROOT\_MODEL** e l’interfaccia **ZSTC\_WEBUI\_IF\_BADI\_MODEL** che porterà i principali metodi di gestione del Model. Anche qui si creerà un Filtro (MODELID) avente come valore l’identificativo definito nella tabella MODEL, per gestire l’associazione della BaDI al relativo Model.

# Programma generatore applicazioni Base

Per facilitare la creazione di applicazioni è stato implementato il report **Zstc\_Webui\_Basic\_App\_Creator.** Il report seguendo le fasi descritte realizza la configurazione del front-end dell’applicazione in pochi secondi fornendo un report dettagliato degli inserimenti per realizzare l’applicazione.

Ecco la screen iniziale del report



Figura 9 - Zstc\_Webui\_Basic\_App\_Creator

Per lanciarlo abbiamo bisogno di alcune operazioni preliminari ovvero creare le strutture nel dictionary per identificare i campi tramite i quali ricercare l’entità, i campi da visualizzare come risultato della ricerca e la sua struttura di dettaglio ovvero cosa mostrare dell’oggetto in dettaglio.

A seguire i passi per realizzare la configurazione front-end

### Strutturazione del dato

Il primo passo da eseguire per realizzare una applicazione è appunto realizzare un disegno di un qualcosa che si voglia rappresentare.

Nel nostro esempio consideriamo la ricerca e la visualizzazione di un cliente, per far ciò creiamo tre strutture.

* Struttura ricerca: ZSTC\_CUSTOMER\_SEARCH\_S
* Struttura risultato: ZSTC\_CUSTOMER\_S
* Tipo Tabella risultato: ZSTC\_CUSTOMER\_T
* Struttura Dettaglio: ZSTC\_CUSTOMER\_DETAIL\_S

Tali strutture conterranno le informazioni per ricercare e rappresentare l’entità cliente, una volta creata l’applicazione comunque si possono sempre aggiungere/togliere campi alle strutture verranno automaticamente letti dall’applicazione

Compiliamo i campi del report citato in precedenza come in figura e lanciamo il report( per scrivere i dati in tabella spuntare la casella modalità di test )



Figura 10 - Zstc\_Webui\_Basic\_App\_Creator

Il log seguente ci mostra che il report è andato a buon fine



Figura 11 - Log esito report

Il primo log contiene il link al quale collegarsi per visualizzare il frontend e tutti gli inserimenti effettuati in tabella. Lanciando il link visualizzeremo appunto l’intera applicazione CLIENTI.

Fino ad ora si è parlato di cosa visualizzare e della sua configurazione, adesso ci occupiamo di come visualizzare il contenuto, naturalmente bisognerà scrivere del codice per recuperare le informazioni desiderate, ma non sono richieste particolari skill di programmazioni ABAP. Si dovrà conoscere principalmente gli enhancement implementatio di SAP, quindi le BAdI e le classi abap.

### Workbench

Prima di tutto recuperiamo il nome degli elementi inseriti in tabella dei quali dobbiamo definirne un comportamento, in particolare i moduli e i model, possiamo recuperarli dal log o andando nelle tabelle elencate sempre nel log.

In particolare per il nostro esempio abbiamo creato i seguenti componenti:

1. Applicazione - Tabella ZSTC\_WEBUI\_APPS
	1. CLIENTI Applicazione Clienti
2. Moduli – Tabella ZSTC\_WEBUI\_MODUL
	1. MOD\_CLIENTI\_ SEARCH Modulo Ricerca
	2. MOD\_CLIENTI\_ DETAIL Modulo Dettaglio
3. Model - Tabella ZSTC\_WEBUI\_MODEL
	1. MODEL\_CLIENTI\_SEARCH Tipo Struttura ZSTC\_CUSTOMER\_SEARCH\_S
	2. MODEL\_CLIENTI\_SEARCH\_RES Tipo Tabella ZSTC\_CUSTOMER\_T
	3. MODEL\_CLIENTI\_DETAIL Tipo Root (Struttura) ZSTC\_CUSTOMER\_S
4. Azioni – Tabella ZSTC\_WEBUI\_ACTN
	1. ACT\_CLIENTI\_SEARCH esegue la ricerca e rimane sul modulo ricerca
	2. ACT\_CLIENTI\_DETAIL legge il dato selezionato e visualizza il dettaglio si sposta sul modulo dettaglio

Dopo aver recuperato i nomi degli elementi di interesse, creiamo un enhancement implementation da se19 utilizzando lo spot **ZSTC\_WEBUI**, per convenzione il nome dovrebbe contenere il nome dell’Applicazione.

N.B Sarebbe conveniente creare un package dedicato per ogni applicazione, dove inserire tutti gli oggetti creati per l’applicazione.



Figura 12 - Enhancement Implementation

Creato l’enhancement implementation si passa a creare le BadI una per ogni Modulo/ Model del quale si voglia specificarne il comportamento nel nostro caso i due moduli e il model del dettaglio.

1. ***Modulo***

Nell’esempio in figura si mostra il Modulo di ricerca, si parte dalla Definizione della BaDI **ZSTC\_WEBUI\_BADI\_MODULE** per i moduli si indica quindi un’implementazione per la badi e il nome della classe di implementazione.



Figura 13 - Creazione BAdI

Dopo aver creato l’implementazione per la BAdI ci spostiamo sulla classe dove andremo a modificare la super classe.



Figura 14 - Classe di implementazione

inserendo la **ZSTC\_WEBUI\_BADI\_ROOT\_MODULE,** nel caso si tratti di un modulo ex novo, se si tratta di un modulo copiato da un'altra app allora inserire la rispettiva classe di implementazione, in modo da lasciare inalterato il suo comportamento di base. In questo modo il nuovo modulo erediterà i metodi e le interfacce e gli attributi della super classe, eventualmente eliminare interfacce duplicate dal tab interfaces.

*N.B Sarebbe meglio prevedere una classe messaggi per applicazione per centralizzare i messaggi.*

Sempre nella classe ridefiniamo il metodo “MAP\_ACTIONS” in modo da aggiungere le due azioni che abbiamo creato in quanto appartengono entrambe al modulo di ricerca.



Figura 15 - Ridefinizione Map\_Actions

Nel nostro esempio abbiamo creato una macro per aggiungere le due azioni, associate ad un token.

La concatenazione del token inserito con il prefisso “DO\_ACTION\_” definiscono il nome del metodo che sarà invocato allo scattare dell’azione corrispondente, da notare che alcuni metodi sono ereditati dalla superclasse quindi basterà ridefinirli per aggiungere del codice, in caso i metodi non esistano si posson creare rispettando sia nomenclatura (DO\_ACTION\_XXX) che la signature e dichiarandoli pubblici.

Nel nostro caso andremo a ridefinire due metodi:

1. Il metodo “DO\_ACTION\_SEARCH” per aggiungere il codice per ricavare i clienti sfruttando i campi definiti nella struttura di ricerca
2. Il metodo “DO\_ACTION\_GET\_DETAIL” per aggiungere il codice che partendo dal record selezionato del model “MODEL\_CLIENTI\_SEARCH\_RES” recuperi tutte le informazioni descritte nella struttura di dettaglio

Attiviamo la classe e torniamo sulla schermata dell’enhancement, per specificare il filtro della BAdI, identificato come la coppia Applicazione (APPID) Modulo (MODULEID) in questo modo ogni azione del frontend sarà gestita da questa BAdI, che si può ampliare a seconda delle azioni inserite nel Modulo



Figura 16 - Filtri per modulo ricerca

1. ***Model***

Per la creazione del model si parte dall’enhancement implementation creando un’ulteriore BAdI per il model come fatto in precedenza, ma partendo dalla definizione BAdI **ZSTC\_WEBUI\_BADI\_MODEL,** invece la classe avrà come super classe la **ZSTC\_WEBUI\_BADI\_ROOT\_MODEL**, anche qui ci saranno i delle interfacce specifiche per i model con i metodi da ridefinire, eventualmente dalla tab interfaces della classe eliminare le interfacce duplicate.

Come per il modulo si deve creare un filtro che leghi la BAdI creata alla tabella MODEL come in figura



Figura 17 - Creazione BAdI Model

Anche per il model definiamo un filtro come per il modulo ma questa volta avremo un solo valore con il nome del Modulo ovvero MODEL\_CLIENTI\_DETAIL



Figura 18 - Filtro BAdI Model

Come fatto per il modulo di ricerca si va a definire una BAdI per il modulo dettaglio, per il nostro caso non è necessaria in quanto andiamo solo a mostrare i dati del cliente ma per un futuro ampliamento.

1. ***Definizione Metodi***

Come accennato nel precedente paragrafo per creare il comportamento dell’applicazione è necessario implementare tre metodi. In particolare due per il modulo di ricerca ed uno sul model:

* 1. ***Modul 🡪DO\_ACTION\_SEARCH***

Metodo del modulo caratterizza l’azione 'ACT\_CLIENTI\_SEARCH'. Qui si andrà a richiamare il model del cliente passandogli le informazioni del model di ricerca di tipo ZSTC\_WebUI\_FWK\_Model\_SF.

Quindi in primis recuperiamo il model di ricerca dall’azione in ingresso riga 8 in figura

Figura 19 - Metodo DO\_ACTION\_SEARCH

Si chiama il metodo del framework per leggere il model di uscita dell’azione riga 10. Tale metodo non fa altro che andare a richiamare l’implementazione del model indicato come id nei parametri. Si consiglia di utilizzare sempre un model radice per gestire la ricerca, in modo tale da poter gestire anche una eventuale lettura gerarchica dei model.

Per verificare i model di I/O controllare la tabella ACTMD.

Recuperato il model di output, ‘MODEL\_CLIENTI\_SEARCH\_RES’ di tipo tabella ZSTC\_WebUI\_FWK\_Model\_T, possiamo restituirlo all’azione. Naturalmente è consigliato gestire la messaggistica per avvisare l’utente sugli esiti delle operazioni. La messaggistica va inserita sempre nell’azione come nel caso in figura riga 18.

* 1. ***Model 🡪SEARCH***

Il metodo in questione viene invocato dal framework method alla riga 10 in figura 19, in particolare va definito all’interno della BAdI del model Clienti dettaglio indicato come radice.

Qui andiamo a leggere il modulo di ricerca per estrapolare i filtri di ricerca tramite il metodo ‘get\_Field\_Values’ del model di ricerca passato in ingresso.

In base ai range ricavati dal metodo eseguiamo una select in tabella per creare una tabella di tipo indicato in selection screen del programma ZSTC\_CUSTOMER\_T

Creiamo un’istanza del model da restituire, model di tipo TABELLA con id MODEL\_CLIENT\_SEARCH\_RES e se il model è stato creato gli impostiamo la base dati estratta. Vedi figura in basso.



Figura 20 - Metodo Model SEARCH

* 1. ***Modul🡪DO\_ACTION\_GET\_DETAIL***

Altro metodo necessario alla nostra applicazione è il get\_Detail, ovvero il metodo associato all’azione ‘ACT\_CLIENTI\_DETAIL’, che scatta alla pressione del button sul widget dei risultati.

Avremo sempre un model di entrata che sarà il record selezionato sulla griglia e uno di output ovvero il modulo clienti dettaglio MODEL\_CLIENTI\_DETAIL, a differenza dell’azione precedente causerà il cambio di modulo, perchè come definito nella tabella ACTN chiamerà il modulo di dettaglio associato ovvero il MODUL\_CLIENTI\_DETAIL.

In questa azione possiamo anche andare a recuperare eventuali model dipendenti per magari associare al Modulo altri widget.

Ad esempio al cliente si possono assocriare le Sales Area dove il cliente è operativo, o i materialimaggiormente richiesti, l’elenco di magazzini e cosi via, per ogni widget naturalmente va gestito un model con la sua struttura dati e una fase di recupero dati.

**Conclusione**

Dopo aver definito la configurazione di customizing e le implementazioni BaDI e le classi sarà già possibile visualizzare il risultato della configurazione di customizing e di workbench collegandosi all’indirizzo dell’applicazione [http://demo.easybp.net:8000/sap/bc/bsp/sap/zstc\_webui\_min?\_APP=](http://demo.easybp.net:8000/sap/bc/bsp/sap/zstc_webui_min?_APP=%20)  CLIENTI

Dove la parte evidenziata i giallo è il nome dell’applicazione BSP e quella in verde il nome dell’applicazione creata. In questo modo l’applicazione BSP provvederà a caricare tutti gli oggetti definiti dal customizing e regolamentati dalla parte di workbench.