Statistical Office of the Republic of Serbia

IST

WHERE INTEGRATION MEETS INFORMATION

IST Developer Guide

Authors:	Mira Nikić Nevena Mitrović
Date:	09.10.2025.
Version	2.0

Contents

1. Introduction 5
1.1 About IST 5
1.2 Purpose of this Guide 6
1.3 Target Audience 6
2. IST Concept and Evolution 6
2.1 What is IST? 6
2.2 Evolution of IST (Past → Present → Future) 7
2.3 Core Principles (Metadata-driven, Standardized, Scalable) 8
3. System Architecture 8
3.1 Metadata Framework 8
3.2 Core Components of IST 9
3.3 Deployment Options (On-premises & Cloud as Server/Storage) 12
4. IST Interpreter 14
4.1 An interpreter for Windows-based environments (.NET) 15
4.2 Web and Android Interpreter (Overview) 16
5. Application Lifecycle in IST 16
5.1 Core Development Workflow in IST 16
5.2 Defining Metadata for Survey Application – Quick start 17
5.3 Designing Forms 19
5.4 Programming Skips, Controls, and Validations 20
5.5 Deployment & Maintenance 27
6. Metadata Database Model (Technical Core) 30
6.1 Metadata Database Structure (IST metadata database model) 30
6.2 Key Metadata Tables - IST Metadata Database 31
6.3 Additional Metadata Tables (Multi-Language Support) 48
6.4 Reserved Words 52
6.5 Performance Best Practices 75
7. Installation and Configuration 80
7.1 Standard Installation 80
7.2 Regional Office Installation 81
7.3 Configuration Details 81
8. Security and Access Control 82
8.1 Authentication & Single Sign-On 82
8.2 User Roles and Permissions 82
8.3 Logging & Auditing 82

9. Integrations 82
9.1 Microsoft Office Integration 82
9.2 OpenOffice / LibreOffice Integration 83
9.3 QGIS (Geospatial Integration) 84
9.4 Power BI (Analytics & Dashboards) 85
10. Appendices 86
10.1 Example Configurations 86
10.2 Detailed Explanation of Functions with Examples 87
11. Developer Troubleshooting and Diagnostics 158
11.1 Common Metadata Errors 158
11.2 Runtime Metadata Validation Tips 158
11.3 Debugging Form Behavior in IST 158
11.4 Useful Logs and Tables for Troubleshooting 158
11.5 SQL Profiler Tips (Advanced) 159
11.6 Best Practices for Debugging Virtual Fields 159
11.7 Tips for Testing in Controlled Environments 159
11.8 Intermittent Errors: Causes and Resolutions 159
Appendix 1 – Full Example of Metadata Setup in IST 160
1. Scenario Overview 160
2. Required Metadata Tables and Example Entries 160
2.1 _IST 160
2.2 _ISTDatabaseConnStrings 160
2.3 _ISTTables 160
2.4 _ISTTableColumns 161
2.5 _ISTRulesDataValidation 162
2.6 _ISTReportsProcedures 162
3. Expected Behavior 162
Appendix 2 – Quick functions summary 163
1. Introduction & Best Practices 163
2. Validations 164
2.1 Range & Type 164
2.2 Length & Patterns 164
2.3 Conditional Required 164
2.4 Multi-Response 165
2.5 Cross-Field & Cross-Table 165
2.6 Warnings (Soft Validations) 165
3. Skips 165
3.1 Basic Skip Model 165

3.2 Multi-Branch & Exceptions....166
3.3 Skip Side-Effects (Disable/Clear)....166
3.4 SkipAction - Common Values and Combinations....166
4. Controls....167
4.1 Visibility....167
4.2 Enable/Disable....167
4.3 Read-Only....167
5. Assignments & Mutations....168
5.1 Assignments....168
5.2 Deletions & Cascades....168
6. Virtual Fields....168
6.1 #FP{} Expressions....168
6.2 SQL-Derived Values....168
7. Batch Rules (Table-Level)....168
8. System Tokens & Special Functions....169
9. Diagrams (Flow & Lifecycle)....170
9.1 Field-Level Validation Lifecycle and Skip Resolution Order....170
9.2 Row level validation (on Save)....171
10. Troubleshooting & Patterns....171
Appendix 3: Copy-Paste Snippets....172
1. Professional Range & Type Check (Age)....172
2. Conditional Required....172
3. Multi-Response Bounds....172
4. Visibility (Show "OtherDetails" when "Other" is chosen)....172
5. Assignment (Initialize Status)....172
6. Batch Rule (SQL condition example)....172
Appendix 4 – Developer Q&A (Frequently Asked Questions)....173
1. Metadata & Application Setup....173
2. Logical Control & Events....173
3. Virtual Fields....173
4. Reports & Output....173
5. Debugging & Testing....173
6. Validation & Logs....174
7. Security & Access....174
8. Web & Mobile....174
9. Performance....174

1. Introduction

1.1 About IST

Introduction

The IST Platform

The Integrated System for Data Processing (IST) is a free, metadata-driven platform developed entirely by the Statistical Office of the Republic of Serbia (SORS) with the aim of modernising and simplifying statistical production.

Designed in line with the principles of the Generic Statistical Business Process Model (GSBPM), IST provides a unified environment for the development and execution of applications supporting statistical surveys and related projects.

The platform is used by:

- IT developers and database specialists – to design and manage applications, metadata, and data flows.
- Statisticians – to collect, validate, and process survey data, as well as to generate dynamic reports (tabulations) and run procedures that control data integrity and consistency.

By integrating application development, data management, and analytical capabilities, IST ensures a streamlined, consistent, and efficient statistical production process.

The platform was built around a simple yet powerful idea:
If data is stored in databases, why not store the application logic in a database as well?

The **IST platform** is built on two core components:

- **Metadata repository - the IST Metadata Database**, which defines every aspect of a statistical application, including variables, forms, logic, validations, and more.
- **Metadata interpreter - the IST Interpreter**, a dynamic engine that reads these metadata definitions and, in real time, generates and executes:
    - Data entry interfaces
    - Validation and control logic
    - Data editing environments
    - Advanced search functions (including cross-survey linking)
    - Reporting and processing procedures

To ensure cross-platform compatibility, IST provides three separate interpreters, all of which operate on the same central metadata database:

- An interpreter for Windows-based environments (desktop, Windows laptop CAPI, CATI),
- A web interpreter for browser-based (CAWI) applications, and
- An Android interpreter for mobile (CAPI) data collection.

This design guarantees consistency across platforms while allowing applications to be executed in diverse operating environments.

In the following sections, we will refer to all three interpreters collectively as the IST Interpreter.

It is important to note that **IST itself does not store or retain statistical data**. All individual and aggregated data related to surveys are stored in separate relational databases, which may reside on one or multiple servers within the NSI infrastructure.

By adopting this model, IST achieves true data and application integration. All information remains in one place, consolidated within a single, user-friendly platform - empowering statistical offices to operate more efficiently, adapt more rapidly, and maintain full control over their data lifecycle.

Again, IST does not store or retain statistical data. It functions exclusively as a metadata-driven platform that dynamically interprets metadata definitions to manage, process, and deliver data-related operations across connected databases.

1.2 Purpose of this Guide

The present document serves as the Developers' Guideline for IST, providing technical instructions specifically intended for use by the IT staff of National Statistical Institutes (NSIs) involved in the design and deployment of survey-based applications.

This document is designed as an introductory tutorial for working with the IST platform. While it provides essential guidance on the most used components and functionalities, it does not cover the full scope of the IST software package.

1.3 Target Audience

This guideline is primarily aimed at: IT staff responsible for the development, configuration, and maintenance of statistical survey applications.

2. IST Concept and Evolution

2.1 What is IST?

The Integrated System for Data Processing (IST) is a unified, metadata-driven platform developed by the Statistical Office of the Republic of Serbia (SORS). In response to the increasing demands placed on National Statistical Institutes (NSIs), IST was created as a long-term solution to simplify and modernize statistical production, reduce redundancies, and improve the quality and efficiency of data handling processes.

Years of experience in statistical systems development led to the design of IST - a platform focused on integrating the entire data lifecycle into a single, coherent environment.

What Does IST Offer?

IST addresses the core challenges of statistical production by:

- Simplifying data processing through standardized workflows,
- Enhancing the value of data via system-wide integration,

- Reducing data and system complexity, enabling better control and understanding.

How Does IST Work?

The IST platform streamlines statistical operations by:

- Bringing data entry, validation, editing, processing, and reporting together in one centralized environment,
- Offering a consistent user interface across all statistical domains, regardless of survey type or methodology,
- Enabling rapid application development and reuse, with seamless data sharing between applications,
- Leveraging existing IT infrastructure, requiring no additional licensing or proprietary software.

What Can IST Do for Your Institution?

Adopting IST can transform your statistical operations by:

- Accelerating survey and application development, reducing time to deployment,
- Integrating data from multiple sources and surveys into a unified system,
- Aligning your processes with GSBPM (Generic Statistical Business Process Model) standards,
- Making metadata actionable in everyday statistical production,
- Enhancing the skills, adaptability, and autonomy of your staff,
- Ultimately, reducing operational costs through automation, reuse, and process optimization.

2.2 Evolution of IST (Past → Present → Future)

The Challenge

In many statistical organizations and IT environments, traditional approaches to application development have led to significant inefficiencies. Each new survey or project often requires a separate, custom-built application, developed in varying technologies - such as Visual Basic, C++, C#, or Java. As a result, data storage and processing become fragmented, ranging from local machines to central servers, and using formats as diverse as relational databases and plain text files.

This technological heterogeneity creates a high degree of vendor and platform dependency, contributing to what can best be described as data chaos. Business processes become sluggish, redundant, and difficult to manage, placing an increasing burden on human resources and slowing the organization's ability to respond to evolving user needs.

Despite the diversity in implementation, almost all data-driven processes - in statistics and beyond - consist of the same three core phases:

- Data Entry
- Data Editing
- Data Processing and Report Generation

Viewed from this perspective, it becomes evident that organizations are repeatedly developing the same logic across different platforms, wasting time and resources.

The Vision

To address these issues, it became clear that what was needed was a unified development environment that:

- Eliminates the need for traditional programming,
- Ensures standardization and uniformity across applications,
- Is resilient to technological change (e.g. new platforms, languages, or tools),
- Brings all data onto a single platform, using a consistent structure (i.e. relational databases).

This vision required a paradigm shift - away from hardcoded applications and toward a metadata-driven approach that would enable full data and process integration, thereby unleashing the full potential of statistical data.

2.3 Core Principles (Metadata-driven, Standardized, Scalable)

The IST platform is structured to be user-friendly and intuitive, allowing IT professionals to focus on what matters most - the data itself. The effectiveness of a statistical survey application built within IST depends greatly on:

- The quality of the underlying database structure,
- The correct definition of metadata,
- The logical accuracy of programmed queries and procedures.

3. System Architecture

3.1 Metadata Framework

IST Architecture and Platform Overview

The Integrated System for Data Processing (IST) operates as a .NET-based interpreter paired with a centralized metadata database. These two equally critical components work together to interact with a wide range of databases containing both individual-level and aggregate data.

At its core, IST reads metadata definitions from its central repository - the IST Metadata Database - which provides a comprehensive description of each application. Using this metadata, IST dynamically generates and executes every stage of a statistical workflow or project in real time, from data entry and validation to data processing and dissemination.

Platform and Operating Environment

- Operating System: IST is developed exclusively for the Windows operating system.
- Database Layer: The IST Metadata Database is hosted on Microsoft SQL Server.
- Framework Dependency: IST is built on the .NET platform and currently requires .NET Framework 4.72 for operation.
- Server Deployment: The program is centrally stored on a dedicated server, from which it is executed across user workstations.

Time Continuity Support

IST supports time-based continuity, enabling the system to display data, applications, and functions dynamically according to the selected time reference. This ensures precise historical tracking and context-sensitive data processing.

Application and Time Point Selection in IST

In IST, the selection of the time point is a fundamental prerequisite for working with metadata. All operations executed within IST - including metadata management, application launch, and data processing - are tied to the currently selected time point.

For certain surveys whose methodology includes non-standard monthly periods (such as the 13th month used in Price Statistics), IST allows for the selection of extended month codes beyond the typical 01–12 range. Specifically, it supports selection from 00 to 99, accommodating diverse temporal structures. For annual surveys, the metadata configuration can omit the month entirely.

Server Directory Structure

All essential files, queries, and scripts are organized on the application server to ensure efficient processing and consistency across modules:

- The core directory path:
  \\serverName\ist\SQLtxt\IST
- Key subdirectories include:
  - ExcelMacro – Templates and macros for generating Excel outputs,
  - Technical Documentation – Documentation and resources for developers and administrators.

This structured approach ensures streamlined operations, seamless maintenance, and full compatibility across the IST ecosystem.

3.2 Core Components of IST

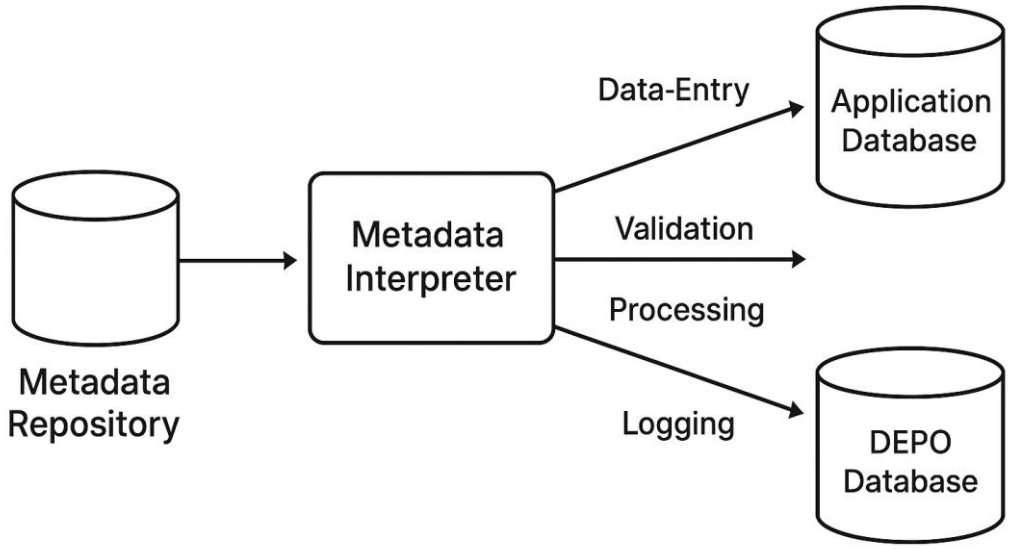


Figure 1: IST System Architecture Overview

The IST platform is built upon a modular architecture, comprising a metadata repository and dedicated interpreters tailored to different modes of data collection.

- Metadata Repository: A simple, centralized metadata database hosted on Microsoft SQL Server, referred to as the IST metadata database, serves as the backbone of the system.
- Metadata Interpreter: A component developed on the .NET Framework, the IST Interpreter reads and executes metadata definitions to enable dynamic survey execution.

There are three types of metadata interpreters, each corresponding to a specific data collection environment:

- Desktop, CAPI (Windows OS), and CATI Interpreter: Supports traditional face-to-face and telephone-based data collection on Windows devices.
- Web Interpreter (CAWI): Enables self-administered online surveys through web browsers.
- Android Interpreter (CAPI on Android OS): Facilitates field data collection using Android-based devices.

All interpreters utilize the same central IST metadata database, ensuring consistency and interoperability across platforms and modes of data collection.

Quality and Architecture of IST Applications

The quality and performance of individual applications developed within IST for data processing are directly dependent on two key factors:

- The quality of the underlying database design, and
- The efficiency and accuracy of the SQL queries and stored procedures that support the application.

Performance and Processing Efficiency

IST is designed to optimize performance by processing one record at a time during data entry and editing, ensuring responsiveness and a smooth user experience. For batch data validation and automatic correction, IST retrieves and processes defined sets of records, allowing efficient batch operations while safeguarding accuracy and consistency across larger datasets.

Modular and Flexible Design

The IST Interpreter is built using a loosely coupled modular architecture. This design principle ensures that individual modules can be easily upgraded, adapted, or extended to meet evolving requirements, without disrupting existing workflows or applications.

During data entry and update processes, the IST Interpreter is optimized to operate on a single record at a time, ensuring maximum speed and responsiveness. Conversely, during batch logical control and automatic correction processes, the system efficiently processes sets of records in bulk, allowing for high-performance validation and correction across large datasets.

Core IST Interpreter Modules

The IST Interpreter* comprises a comprehensive suite of modules that cover every stage of the data processing lifecycle:

- Metadata Maintenance Module – Management and updating of metadata structures.
- Real-Time Generated Data Entry (Web, desktop, android)– Dynamic creation of data entry forms for efficient collection.
- Batch Logical Control – Automated validation processes applied to large datasets.
- Data Checking and Validation – Ensures consistency and accuracy of entered data.
- Data Editing and Correction – Tools for manual and automated data adjustments.
- Batch Automatic Correction – Automated error detection and correction procedures.
- Advanced Search – Enables in-depth querying of both individual and aggregated datasets across multiple databases.
- Report Generation – Production of outputs in various formats, including Excel, JSON, and XML.
- Data Management – Centralized control of data storage and structures.
- Procedure Management – Execution and oversight of SQL procedures for both individual-level and aggregate data.

This modular approach ensures that IST remains scalable, efficient, and adaptable, supporting both current operations and future enhancements in statistical production.

Multi-Instance Capability

IST is designed to support multiple concurrent instances. This enables users to work on several applications simultaneously. Each running instance operates independently on its own:

- Dedicated portion of RAM,
- Separate connection to the metadata database, and
- Self-managed connection lifecycle, where every request to the database opens and closes its connection automatically.

This design ensures reliable, high-performance operation and allows seamless concurrent work across different applications and processes.

Error Reporting, Forensics, and Audit Trail

IST maintains a comprehensive logging and audit framework to ensure transparency, accountability, and full traceability of all validation, correction, and user actions.

Deletion Forensics and Audit Trail

- _ISTLogDelete – Every deletion of a record in IST interpreter automatically creates an entry in this table within the DEPO database.
- Logged details include: survey name, table name, user account, date, time, and the primary key of the deleted row.
- Only INSERT actions are permitted in this table; deletions or modifications are strictly prohibited.

Batch Logical Control Logging

- _ISTLogBatchLC – Every execution of batch logical control (validation across a set of records) automatically generates a log entry in this table within the DEPO database.
- Each log provides traceability for validation checks performed during batch operations.

Variable Change Logging

- _ISTVariablesChange – By clicking the Save button from the data entry or data editing form, IST records all edits and inputs of variables in this table.
- Logged data captures which variable was changed, its new value, and the user who performed the change.

DEPO Database

- All three logging tables (_ISTLogDelete, _ISTLogBatchLC, _ISTVariablesChange) are stored in the DEPO database located on the MS SQL Server.
- The DEPO database is publicly accessible to all IST user accounts.
- For security and integrity:
    - Only INSERT operations are allowed.
    - Physical deletion of these tables is impossible without administrative permission.

Audit and Usage Logs

In addition to forensics logs, IST provides audit trails for dissemination and process usage:

- _ISTLogProcessUsage – Records every process initiated in IST, including report name, parameters used, execution timestamp, and user.
- _ISTLogExcelXMLJSON – Stores the underlying SQL query behind each dataset exported (Excel, XML, JSON).
- _ISTSavedAdvancedSearch – Saves the query behind each dataset exported through Advanced Search, ensuring reproducibility.

These logs provide end-to-end traceability and support compliance with GSBPM 5.1 Alignment by documenting key steps in data dissemination and user interaction.

3.3 Deployment Options (On-premises & Cloud as Server/Storage)

The IST platform does not require a traditional installation process. To access IST, users only need to:

- Create a shortcut on their workstation pointing to the ProcessStarter.exe file located on the server (\\serverName\ist), or
- Simply copy the existing IST.lnk shortcut from the server to their local PC.

Version Management

To ensure uninterrupted production operations, IST is not launched directly from its executable file. Instead, it is started through ProcessStarter.exe, a utility designed to manage and switch between different IST versions when required.

The version of IST being executed is controlled by the configuration file ProcessStarter.exe.config, which specifies the path to the current active version. The system maintains up to six previous IST versions, stored sequentially as IST1, IST2, ..., IST6... allowing for seamless rollbacks or parallel testing when necessary.

Configuration Example

Below is an example of the configuration file:

```
<?xml version="1.0" encoding="Windows-1252"?>
<configuration>
 <appSettings>
  <add key="AppPath" value="\\serverName\IST\IST4.NET\Istrazivanja.exe"/>
 </appSettings>
</configuration>
```

This setup ensures that IST can be updated or modified at any time without interfering with the stability or availability of the current production environment.

IST Configuration Data File

The configuration file Istrazivanja.exe.config (***Anex 1***) is an XML-based document that defines key settings required for the proper functioning of the IST .NET Interpreter. Its primary function is to specify the connection string to the IST Metadata Database, allowing the interpreter to establish secure communication and perform all necessary operations.

Structure and Sections

The configuration file consists of multiple sections, each serving a specific purpose in defining connections and operational parameters for IST components:

- ISTConnection – Provides the connection string for the IST metadata database.
- ISTDeletings – Defines parameters for managing and storing keys of deleted records in the DEPO database.
- ISTCATI – Specifies the configuration for CATI (Computer-Assisted Telephone Interviewing) operations.
- ISTCAPI – Provides settings for CAPI (Computer-Assisted Personal Interviewing) operations.
- Istrazivanja – Includes additional operational and user interface parameters for IST applications.

Key Notes

- Connection strings should always point to the correct SQL Server instance and database names.
- User credentials and permissions must be managed securely and updated as required by institutional IT policies.
- Paths to resources such as input programs, help files, and technical documentation must be maintained accurately to ensure smooth operation of IST applications.

4. IST Interpreter

How the IST .NET Interpreter Works

When an application is selected, the IST .NET interpreter loads metadata from the IST metadata database into RAM. This includes all relevant metadata tables for the chosen application and time period.

Initialization

- On startup, IST .NET automatically sets the last survey and time point used by the user.
- These values are stored in the Windows Registry (HKEY_CURRENT_USER/Software/VB and VBA Program Settings/Istrazivanja) and reloaded at each session.
- Each IST instance opens a new SQL Server connection to the metadata database and reserves its own RAM allocation.

Real-Time Form Generation

- Clicking DataEntry triggers the on-demand generation of the data entry form.
- At the start, no form exists - IST builds it dynamically from metadata.
- Fields, validation logic, and events are placed on an empty canvas in real time, based on definitions in _ISTTablesColumns and related metadata tables.

Parallel Operation

- Every selected app establishes its own connection to the subject-matter database on SQL Server.
- Multiple IST applications can run simultaneously, processing different surveys or the same survey at different time points.
- All connections are automatically closed when applications terminate.

Metadata Loaded into RAM

The following elements are preloaded to RAM for performance:

- ComboBox datasets
- Rules for skips, validation (type, length, allowed values)
- Rules for enabling/disabling controls and subtables
- Events and actions on entry/exit of controls
- Virtual field definitions (computed fields not stored in the database)
- Rules related to data editing

Form generation speed depends only on the number of fields defined in _ISTTablesColumns. There are no restrictions on field count.

Data Entry and Record Handling

- Once the form opens, IST uses the primary key values to connect to the database and retrieve the relevant record.
- That record is cached in RAM as a dataset for fast access.
- IST works at the single record level during data entry, ensuring high responsiveness.

Application Behavior

- IST applications are Windows Forms–based and fully support Unicode.
- Partial saving is supported: users may save incomplete records and continue later.
- Data is written to the database either when the Save button is pressed (on Master or Detail forms) or when navigating from a Master form to its associated Detail form
- Entering a Detail row saves the Master row to RAM and to the database.

Authentication Models

- CAPI and CATI modules use login forms for user identification and authentication
- Desktop module uses Single Sign-On (SSO)
- Web interpreter must be integrated into NSO's existing web authentication system

Default Behaviour and Performance Considerations

By default, IST performs logical control (validation and data checking) whenever the user exits a field. If validation rules involve multiple cross-table/database checks, or if many virtual fields depend on queries, this can significantly slow down performance.

4.1 An interpreter for Windows-based environments (.NET)

(desktop, Windows laptop CAPI, CATI)

4.1.1 Active Domain Users

To use the IST Desktop version, it is necessary for all users to be registered as Active Directory Domain Users within the institutional IT infrastructure. This configuration enables secure authentication and ensures proper access control to the IST platform and underlying SQL Server databases, in accordance with enterprise-level security policies.

IST relies on Windows Authentication to manage secure access to SQL Server databases. All applications developed within the IST framework adhere to the defined user roles, permissions, and authorization structures established on the corresponding SQL Server instance. This ensures that access control and data security are consistently enforced in accordance with institutional IT policies.

4.1.2 Regional Office Users

Installation of IST in Regional Offices

For regional offices, where IST is not launched directly from the central server, installation is streamlined to minimize network and server load.

If the IST Interpreter is not executed directly from the server, it is sufficient to copy the Ist*N*.NET directory to the user's local workstation. This approach ensures that the local network and application server remain unburdened, while still providing full functionality to regional users.

Automatic Version Control

To support these deployments, an auxiliary program named ISTCheckThanStart.exe (located on the server in the same directory as ProcessStarter.exe) has been developed. This program ensures that users always run the latest available version of IST by:

- Checking the current version on the user's workstation,
- Automatically copying the latest version from the server to the local machine if an update is required, and
- Storing the local version in the %Temp%\IST directory (creating this directory if it does not exist).

If the local installation is already up to date, no copying is performed, ensuring fast and efficient startup.

4.2 Web and Android Interpreter (Overview)

The IST Interpreter for Web and Android platforms enables the execution of survey logic and validations directly within browsers or Android devices, using the same central metadata database as the desktop version. Although they rely on a unified metadata structure, the Web and Android interpreters are technically distinct, tailored to the specific requirements of their platforms - ensuring smooth, responsive performance in web environments and offline functionality on mobile devices. Importantly, not all functionalities available in the desktop interpreter are implemented here; only those features that are suitable and technically feasible for browser-based and mobile operation are supported. This selective implementation ensures optimal usability and performance across devices, while maintaining full compatibility with core IST survey definitions, validations, skip logic, and user interface metadata. As a result, field data collection and review can be carried out efficiently, without compromising essential survey logic.

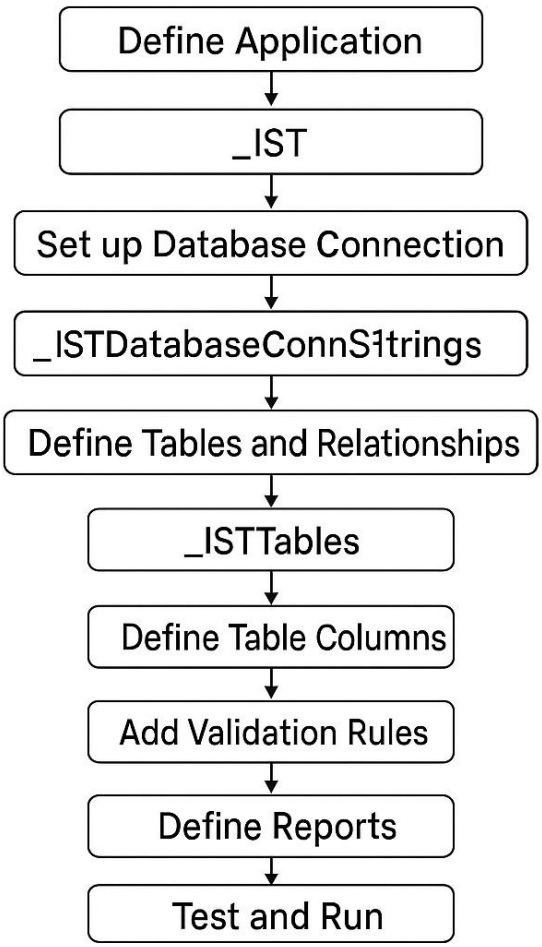
5. Application Lifecycle in IST

5.1 Core Development Workflow in IST

The following activities outline the typical development cycle when working with IST:

- Metadata Definition: Creating a survey dictionary that includes question content, variable names, data types, and structural definitions.
- Code List Creation: Defining valid responses to each question (e.g., for the question on sex: 1 = Male, 2 = Female).
- Form Design: Structuring the digital questionnaire interface.
- Logic Programming: Implementing skip patterns, validation rules, and other questionnaire logic.
- Application Deployment: Installing the finalized application onto laptops, tablet PCs, or similar field devices.

5.2 Defining Metadata for Survey Application – Quick start



Quick Start: Your First IST Survey Application

Get started building and running your first survey application in IST in minutes. This step-by-step guide walks you through the essential metadata setup required to launch a basic survey application.

Step 1: Define Your Application Metadata in the _IST Table

Open the IST metadata database and insert a new row into the _IST table with the following fields:

Field	Example Value
appCode	DAPP
appTitle	Demo – Household Survey
databaseAlias	DDATA
additionalParameters	
ValidFrom	2025-01-01
ValidTo	2025-12-31

Step 2: Set Up Database Connection in _ISTDatabaseConnStrings

Insert a row into the _ISTDatabaseConnStrings table to define the connection string for your data database:

Field	Example Value

databaseAlias	DDATA
ODBC	Data Source=localhost;Initial Catalog=DemoDB;Integrated Security=SSPI;
ValidFrom	2025-01-01
ValidTo	NULL

Step 3: Define Tables and Relationships in _ISTTables

Define your master and child tables in the _ISTTables metadata table:

appCode	tableName	typeOfTableParentChild	parentTableName	tableDescription	ValidFrom
DAPP	Households	G		Households	2025-01-01
DAPP	Members	D	Households	Members	2025-01-01

Step 4: Define Table Columns in _ISTTableColumns

Define fields for each table, including primary keys, data types, labels, and order:

appCode	tableName	columnName	primary Key	column Type	column Length	label	order Number	columnAttributes
DAPP	Households	HHID	P	int	4	Household ID	1	
DAPP	Members	HHID	P	int	4	Household ID (FK)	1	
DAPP	Members	MemberID	P	int	4	Member ID	2	
DAPP	Members	Age		int	3	Age	3	
DAPP	Members	IsAdult				Is Adult?	4	case when isnull(Age,0)>=18 the 1 else 0 end

IsAdult is virtual field calculated in run time and does not exist in Members table of corresponding DDATA database

Step 5: Add Validation Rules in _ISTRulesDataValidation

Define rules for data validation in the _ISTRulesDataValidation table:

appCode	tableName	errNumber	Error	condition	errTitle	errWeight	ValidFrom
DAPP	dbo.Members	E001	Age < 0 OR Age > 120	ExecuteOnlyOnDataEntryForm	Age must be between 0 and 120	T	2025-01-01

Field E001 must be defined in the Members table of the corresponding DDATA database.
The data type of all error fields, including E001, must be bit.
The field E001 will be assigned the value True (1) if the condition Age < 0 OR Age > 120 is met, and False (0) otherwise.
The condition ExecuteOnlyOnDataEntryForm specifies that this validation rule will be executed only during data entry or data editing and will not run during batch logical control (batch mode validation).

Step 6: Define Reports in _ISTReportsProcedures

Define reports using SQL queries in the _ISTReportsProcedures table:

appCode	SeqNumber	Title	query	ValidFrom
DAPP	1	Members Summary	select HHID, count(*) as TotalMembers, avg(Age) as AvgAge from dbo.Members group by HHID	2025-01-01

Step 7: Test and Run Your Application

Enter data for households and members. Verify that validation and reports work as expected. The application should allow entry of households and their members, automatically validate member age, compute the virtual field 'IsAdult' based on age, and generate a summary report grouped by household.

5.3 Designing Forms

1. Form Generation from Metadata

Forms in IST are generated dynamically based on metadata definitions. The following tables control form structure and behavior:

- _ISTTables – defines which tables appear on the form and their relationships.
- _ISTTableColumns – defines fields, their order, labels, control types, and attributes.
- _ISTLabels – provides multilingual labels for fields and forms.

2. Layout Control

Form layout is controlled by metadata attributes such as:

- orderNumber – determines the sequence of fields on the form.
- NewQuestionGroupButtonLocation – used to visually group related fields.
- typeOfTableParentChild – defines parent-child relationships for hierarchical forms.

3. Field Properties

Each field in a form can be configured with:

- Control type – specified in columnAttributes (e.g., DataGridView, AutoCompleteTextBox).
- Mandatory – fields marked as required will be validated automatically.
- Virtual fields – calculated fields using formulas (e.g., #FP{IsAdult}=case when isnull(d.Age,0)>=18 then 1 else 0 end).

4. Validation and Logic

Validation rules and logic are defined in metadata:

- validatingEvent in _ISTTableColumns for field-level checks.

- _ISTRulesLogicalControl for cross-field logic and automatic corrections.

5. Best Practices

Recommendations for designing effective forms:

- Keep forms simple and intuitive - avoid overcrowding with too many fields.
- Use AutoCompleteTextBox for large lists to improve performance.
- Apply multilingual labels via _ISTLabels for internationalization.
- Preview forms in the IST Interpreter to validate layout and behavior.

6. Sample Metadata Snippet

Below is an example of how a form layout is defined in _ISTTableColumns:

appCode: DEMOAPP
tableName:Members
columnName: Age
orderNumber: 3
label: Age
columnType: int
columnLength: 3
validatingEvent: stopIf= isnull(d.Age,0)<18;StopMsg=Age must be greater or equal than 18
columnAttributes: for #FP{IsAdult}: case when isnull(d.Age,0)>=18 then 1 else 0 end

5.4 Programming Skips, Controls, and Validations

Introduction

In IST, programming skips, controls, and validations is entirely metadata driven. This approach enables rapid, no-code configuration of complex survey logic, robust data validation, and dynamic user interface behavior - without traditional programming. All logic is defined in metadata tables and interpreted at runtime, ensuring consistency, maintainability, and scalability across desktop, web, and mobile survey applications.

Core Concepts

- Skips: Conditional navigation - automatically moves focus to another field or section based on user input.
- Controls: Dynamic UI behaviors - enabling/disabling, showing/hiding, or making fields read-only.
- Validations: Rules that ensure data integrity - ranging from simple type checks to complex cross-field logic.
- All are defined declaratively in IST metadata tables, primarily ISTTablesColumns and ISTRulesDataValidation.

Programming Skips (Skip Logic Syntax)

Skips are defined using the SkipTo and SkipIf parameters in the validatingEvent field of ISTTablesColumns:

#{
 SkipTo=TargetField1$TargetField2;

```
 SkipIf=Condition1$Condition2;
 SkippedEnabledFalse;
 SkippedSetEmpty;
}
```

SkipTo: Comma- or $-separated list of destination controls.
SkipIf: Boolean expressions; each maps to the corresponding SkipTo.
SkippedEnabledFalse: Disables skipped controls.
SkippedSetEmpty: Clears skipped controls.

Example 1: Simple Skip

```
#{
 SkipTo=Q41;
 SkipIf=4 not in Pomocno_G4;
 SkippedEnabledFalse;
 SkippedSetEmpty;
}
```

If value 4 is not in Pomocno_G4, focus jumps to Q41, and skipped fields are disabled and cleared.

Example 2: Multi-branch Skip

```
#{

 SkipTo=Q41$Q37_7;
 SkipIf=6 not in Pomocno_G4$6 in Pomocno_G4;
 SkippedEnabledFalse;
 SkippedSetEmpty;
}
```

Two branches: if 6 is not in Pomocno_G4, skip to Q41; if it is, skip to Q37_7.

Example 3: Skipping with Exceptions

```
#{
 SkipTo=BrojDom$UBrLicaStan$ButtonNext;
 SkipIf=cast(isnull(KoZiviuStanu,0) as int)=1$cast(isnull(KoZiviuStanu,0) as
int)=2$cast(isnull(KoZiviuStanu,0) as int) in (3,4,5,6,7);
 SkippedEnabledFalse;
 SkippedSetEmptyExcept=BrojDom$BrojDom,UBrLicaStan;
 SkippedSetEmptyExceptIf=cast(isnull(KoZiviuStanu,0) as int)=2$cast(isnull(KoZiviuStanu,0) as int) in
(3,4,5,6,7);
}
```

Skips and disables fields, but keeps BrojDom and UBrLicaStan under certain conditions.

Programming Controls (Dynamic UI Behavior)

Controls are managed using parameters like ReadOnly, VisibleFalse, EnabledFalse, and their conditional counterparts (ReadOnlyIf, etc.):

```
#{
```

```
 ReadOnly=field1,field2;
 ReadOnlyIf=isnull(flag,0)=1;
 VisibleFalse=field3;
 VisibleFalseIf=isnull(hide,0)=1;
}
```

Example 4: Conditional Read-Only

```
#{
 ReadOnly=MBR,OPS;
 ReadOnlyIf=cast(isnull(IsImported,0) as int)=1;
}
```

Makes MBR and OPS fields read-only if IsImported is set.

Example 5: Dynamic Visibility

```
#{
 VisibleTrue=ReasonText;
 VisibleTrueIf=cast(isnull(HasReason,0) as int)=1;
}
```

Shows ReasonText only if HasReason is true.

Example 6: Enabling/Disabling

```
#{
 EnableFalse=AllInputs;
 EnableFalseIf=cast(isnull(IsSubmitted,0) as int)=1;
}
```

Disables all input fields if the record is marked as submitted.

Programming Validations

Field-Level Validation

Validation rules can be defined directly in the validatingEvent field for immediate feedback:

```
#{
 Min=1; Max=31; Yes=numeric; MnLength=2; MxLength=2;
 StopIf=isnull(d.Age,0)<18;
 StopMsg=Age must be greater or equal than 18;
}
```

Checks that Age is between 1 and 31, is numeric, exactly 2 digits, and at least 18.

Table-Level (Batch) Validation

Batch rules are defined in ISTRulesDataValidation:

appCode	tableName	errNumber	Error	condition	errTitle	errWeight	ValidFrom
DAPP	dbo.Members	E001	Age < 0 OR Age > 120	ExecuteOnlyOnDataEntryForm	Age must be 0–120	T	2025-01-01

This rule marks records with invalid ages and highlights the error in red.

Multi-Response and Complex Patterns

Example 7: Multi-Response Numeric

```
#{
 Yes=MultiResponseNumeric; Min=1; Max=12;
 MnResponseCount=5; MxResponseCount=5;
}
```

Requires exactly 5 numeric responses between 1 and 12.

Example 8: Hard Validation with Stop

```
#{
 StopIf=isnull(ans,'') like '%3%' and (isnull(ans,'') like '%1%' or isnull(ans,'') like '%2%');
 StopMsg='Category 3 cannot be mixed with 1 or 2';
}
```

Blocks saving if category 3 is selected together with 1 or 2.

Advanced Examples

Assignments and Auto-Fill

```
#{
 AssignTo=Status;
 AssignWhat='NEW';
 AssignIf=isnull(Status,'')='';
}
```

Sets Status to 'NEW' if empty.

Conditional Deletion

```
#{
 deleteFrom=Trace1,Trace2,Lica;
 deleteWhere=rbr>cast(#FP{fprbr} as integer);
 deleteIF=#FP{fprbrTRACE1}>#FP{fprbr};
}
```

Deletes rows in child tables if the roster size shrinks.

Virtual Fields

Virtual fields are defined in columnAttributes and calculated at runtime:
#FP{IsAdult}=case when isnull(d.Age,0)>=18 then 1 else 0 end
Calculates if a member is an adult based on age.

Best Practices

- Keep logic in metadata: Avoid hardcoding; use metadata for all skips, controls, and validations.
- Use virtual fields for derived values: Reduces redundancy and improves maintainability.
- Prefer field-level validation for performance: Use StopIf and SkipIf for immediate feedback.
- Batch validation for cross-record checks: Use ISTRulesDataValidation for rules that require context.
- Test with realistic data: Use the IST interpreter's preview and batch validation modules.
- Document complex rules: Use comments in metadata or maintain a separate logic map.

Troubleshooting Tips

- Field not visible? Check ISTTablesColumns and time validity.
- Validation not firing? Ensure logic is in the correct field and event.
- ComboBox empty? Check relational table/column definitions.
- Virtual field blank? Test the expression in SQL Server Management Studio.

Minimal Metadata Example

Household Survey Example
Tables: Households (master), Members (child)
Virtual Field: IsAdult = Age >= 18
Validation: Age must be 0–120
Skip: If IsAdult=0, skip employment questions

```
#{
 SkipTo=NextSection;
 SkipIf=IsAdult=0;
 SkippedEnabledFalse;
 SkippedSetEmpty;
}
```

Practical Validation Examples for IST

Range and Type Checks

Numeric Range (Age 0–120):

```
#{
 Min=0; Max=120; Yes=numeric;
 StopIf=isnull(Age,0)<0 or isnull(Age,0)>120;
 StopMsg='Age must be between 0 and 120';
}
```

Note:
You can use:

Option 1: Min=0; Max=120
Option 2:
StopIf=isnull(Age,0)<0 or isnull(Age,0)>120;
StopMsg='Age must be between 0 and 120';
Option 3: Combine both approaches for stricter validation and differentiated messages.

Text Length (ID must be 13 digits):

```
#{
 MnLength=13; MxLength=13; Yes=numeric;
 StopIf=len(isnull(JMBG,''))<>13;
 StopMsg='JMBG must be exactly 13 digits';
}
```

Date Format (must be dd.MM.yyyy):

```
#{
 Yes=istDate;
 StopIf=isnull(DateField,'')<>'' and isDate(DateField)=0;
 StopMsg='Date must be in format dd.MM.yyyy';
}
```

Conditional Required Fields

Field required if another field has a value:

```
#{
 RequiredIf=Status='Active';
 StopIf=isnull(ActiveDate,'')='';
 StopMsg='Active Date is required when Status is Active';
}
```

At least one of two fields must be filled:

```
#{
 StopIf=isnull(Phone,'')='' and isnull(Email,'')='';
 StopMsg='Either Phone or Email must be provided';
}
```

Allowed/Disallowed Values

Allowed values (Gender):

```
#{
 Yes=1,2; // 1=Male, 2=Female
 StopIf=isnull(Gender,'') not in ('1','2');
 StopMsg='Gender must be 1 (Male) or 2 (Female)';
}
```

Disallowed value (Code 999 not allowed):

```
#{
 No=999;
```

```
 StopIf=Code=999;
 StopMsg='Code 999 is not allowed';
}
```

Multi-Response Validations

Multi-Response Numeric (max 3, min 1):

```
#{
 Yes=MultiResponseNumeric; Min=1; Max=10;
 MnResponseCount=1; MxResponseCount=3;
 StopIf=ResponseCount<1 or ResponseCount>3;
 StopMsg='Select between 1 and 3 options';
}
```

Disallow certain combinations:

```
#{
 StopIf=isnull(Answers,'') like '%3%' and (isnull(Answers,'') like '%1%' or isnull(Answers,'') like '%2%');
 StopMsg='Option 3 cannot be combined with 1 or 2';
}
```

Cross-Field Validations

End date must be after start date:

```
#{
 StopIf=EndDate<StartDate;
 StopMsg='End date must be after start date';
}
```

If "Other" is selected, specify details:

```
#{
 StopIf=Option='Other' and isnull(OtherDetails,'')='';
 StopMsg='Please specify details when "Other" is selected';
}
```

Custom SQL/Function Validations

Check if value exists in a reference table:

```
#{
 StopIf=not exists(select 1 from RefTable where RefCode=InputCode);
 StopMsg='Input code does not exist in reference table';
}
```

Check modulo 11 for registration number:

```
#{
 StopIf=#MOD11{RegNumber,8}=0;
 StopMsg='Registration number is not valid (modulo 11 check failed)';
}
```

Soft Validation (Warning Only)

Show warning but allow save:

```
#{
 Warning;
 Min=0; Max=100;
 SimpleMsgIs='Value is outside the typical range, please check.';
 SimpleMsgIf=Value<10 or Value>90;
}
```

Validation with Messages from ISTMessages Table

Centralized message:

```
#{
 StopIf=isnull(PIN,'')='';
 StopMsg=ISTMSG050; // e.g., 'PIN is required'
}
```

Validation on Save (Batch/Record Level)

Record must be unique (no duplicate key):

```
#{
 StopIf=exists(select 1 from Table where Key1=d.Key1 and Key2=d.Key2 and ID<>d.ID);
 StopMsg='Duplicate record: combination of Key1 and Key2 must be unique';
}
```

Advanced: Virtual Field Validation

Virtual field must be positive:

```
#{
 StopIf=#FP{TotalAmount}<=0;
 StopMsg='Total amount must be positive';
}
```

5.5 Deployment & Maintenance

The deployment and maintenance phase in the IST ecosystem ensures that developed applications are successfully installed, versioned, monitored, and supported in both local and cloud environments. Because IST is a metadata-driven platform, deployment focuses less on compiling software and more on ensuring that the correct metadata, configuration files, and SQL objects are synchronized between development, test, and production environments.

Deployment Architecture

IST supports both on-premise and cloud-based (Azure) deployments.

Each environment consists of three key components:

1. IST Interpreter (.NET Application) – stored on the shared network path (e.g. \\serverName\IST) and accessed via the ProcessStarter.exe launcher.
2. IST Metadata Database – hosted on Microsoft SQL Server; contains configuration, validation rules, forms, and reports.

3. Subject Matter Databases – individual or aggregated data repositories referenced through the _ISTDatabaseConnStrings table.

To simplify deployment across regional offices and departments, IST applications are versioned (e.g. IST1–IST6) allowing administrators to roll out updates gradually without interrupting production work.

Version Management

Each IST instance is launched through ProcessStarter.exe, which dynamically points to the latest approved version of the IST interpreter defined in ProcessStarter.exe.config.

This setup provides the following benefits:

- Zero downtime updates – new versions can be tested independently before being made active.
- Backward compatibility – older versions remain available in parallel for validation or rollback.
- Automatic propagation – via ISTCheckThanStart.exe, ensuring all regional offices automatically receive the latest executable when starting IST.

The configuration file also maintains connection strings for different modules (CAPI, CATI, CAWI, DEPO), ensuring that each environment can operate autonomously with minimal user configuration.

Maintenance and Monitoring

The IST framework includes built-in logging and version control mechanisms that simplify long-term maintenance:

- Operational Logs: All user actions such as deletions, logical controls, and report executions are automatically written to the DEPO database.
- Performance Monitoring: Developers should regularly review execution times of SQL rules and optimize slow queries identified via metadata or DEPO logs.
- Backup Policy: The \SQLtxt, \ExcelMacro, and \TechnicalDocumentation directories on the IST file server are subject to regular backups, ensuring version safety.
- Metadata Evolution: When survey methodologies or variables change, developers should duplicate metadata entries with updated ValidFrom dates instead of overwriting existing records.

Cloud Deployment (IST on Azure)

When deployed in the cloud, IST leverages Microsoft Azure services to host both metadata and data servers.
Key benefits include:

- Scalability: Dynamic resource allocation during large-scale survey operations.
- Collaboration: Multi-office teams can access shared applications in real time.
- Reduced maintenance costs: No local server hardware or manual patching required.

Developers deploying IST to Azure should ensure the following:

- Metadata and data connection strings use Azure SQL endpoints.
- Blob storage paths replace local file references for reports and macros.
- Access rights are synchronized with Azure Active Directory to maintain secure SSO authentication.

Maintenance Best Practices

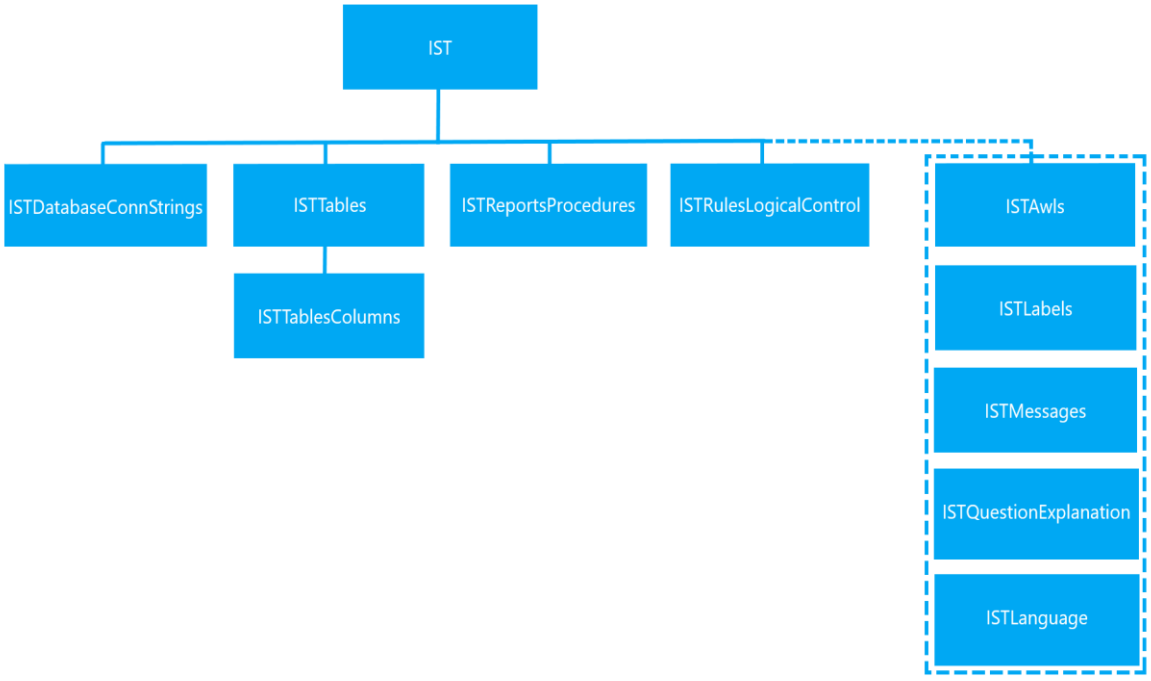
To keep the system stable and secure, developers should:

- Regularly check SQL connections defined in _ISTDatabaseConnStrings.
- Validate that all .SQL, .XML, and .XLSM files referenced in metadata exist on the shared server path.
- Clean obsolete temporary folders in %Temp%\IST.
- Maintain strict naming conventions for applications and tables to prevent reference conflicts.
- Review metadata relationships periodically, especially inherited apps defined in the additionalParameters field of _IST metadata table.

6. Metadata Database Model (Technical Core)

6.1 Metadata Database Structure (IST metadata database model)

IST metadata is composed of the following tables:



IST Metadata Structure

The IST Metadata is implemented as a relational database, serving as the foundation for defining, managing, and controlling applications developed within the IST platform. This metadata layer provides all necessary instructions for the creation, execution, and maintenance of statistical surveys and related processes.

Core Metadata Tables

Table Name	Description
_IST	Contains the core description of each application developed within IST.
_ISTDatabaseConnStrings	Stores the connection strings for databases containing both individual and aggregated (subject-matter) data.
_ISTTables	Defines relational tables that store statistical survey data and the relationships between those tables.
_ISTTablesColumns	Provides detailed definitions of table fields listed in _ISTTables, including sets of rules that the IST.NET interpreter applies during data entry.
_ISTRulesLogicalControl	Contains data editing and automatic correction rules specific to a statistical survey.

Table Name	Description
_ISTReportsProcedures	Stores report definitions. Each report is a combination of an Excel file (XLS, .XLSX, .XLSM), an XML file, and one or more SQL queries or stored procedures used to generate and format output.

Time-Driven Metadata

All IST metadata tables include the columns VALIDFROM and VALIDTO, which define the time period during which the record is active.

When a time point (month and year) is selected within IST, the system retrieves only the metadata records that are valid for that period. This feature enables statisticians to:

- Configure different application versions for data entry and editing,
- Apply time-specific rules,
- Ensure consistent alignment of metadata with evolving survey requirements over time.

Reserved Words

Certain reserved words are used within the IST metadata and in databases containing individual-level data. These reserved words should be used consistently across all applications to ensure uniformity and proper system interpretation.

Reserved Word	Meaning
GOD or ISTYEAR	Year
MES or ISTMONTH	Month
KV	Quarter

Important Distinction

It is essential to distinguish between data and metadata references throughout the IST environment. When referencing a database, table, column, or row:

- *Data refers to relational datasets containing individual-level records collected through statistical surveys.*
- *Metadata refers to structural and descriptive information stored in the IST Metadata Database, which drives the design, execution, and management of these applications.*

6.2 Key Metadata Tables - IST Metadata Database

_IST – Application Description Table

The _IST table is the core metadata table in the IST platform. It defines and controls the general configuration of each application built within IST. This table acts as the foundation for linking metadata components, ensuring that applications operate consistently across all modules and processes.

Purpose

The _IST table serves as the central descriptor of an IST application. It contains essential information such as the application code, title, database references, server configurations, validity periods, and technical paths to queries, macros, and external entry programs.

By maintaining these definitions centrally, the _IST table ensures:

- Seamless integration between the IST interpreter and relational databases.
- Accurate execution of application-specific processes such as data entry, editing, validation, reporting, and dissemination.
- Time-based versioning to support updates and maintain historical integrity.

Key Fields and Descriptions

pk primary key field
m mandatory field
– field is not in use

Field	Type		Description
appCode	nvarchar (8)	pk	Code of the application An unique identifier for each application. Note: characters /, * should not be used
appInEditMeta	int		Indicates whether the application can be edited through the graphical user interface (GUI). Applications created via the GUI have a default value of 4.
appTitle	nvarchar (255)	m	The official name or title of the application.
SaveExecOnLoadParameters	nvarchar (255)		A multipurpose field controlling specific runtime behaviors, such as: SaveRecordsImmediately – Automatically saves data in parent-child table structures without manual confirmation (Default) ExecOnLoad – Executes a specified stored procedure upon application startup, typically for CAPI data collection. Multiple parameters can be combined and separated by semicolons (;).
CAPIServerName	nvarchar (255)		The name and IP address of the server used for CAPI (Computer-Assisted Personal Interviewing) operations. Both values are stored together, separated by the $ sign (e.g., serverName$XX.XX.XX.XX).
Periodics	nvarchar (24)		Defines the periodicity of the survey (e.g., monthly, quarterly, semi-annual). This field is also used by web interpreter to configure period-selection dropdowns in the web interface.

tablesNumber_NA	int	-	Not used
databaseAlias	nvarchar (10)		A logical name of the relational database connected to the application. The corresponding connection string is stored in the _ISTDatabaseConnStrings table.
additionalParameters	nvarchar (1255)		Supports metadata inheritance from other applications. This feature reduces duplication by allowing one application to inherit tables, fields, rules, and report procedures from another, while adding only the metadata required for customization. ***
pathToQuery	nvarchar (255)		Path to the directory where SQL scripts, Excel macros, headings, and other supporting files are stored (e.g., \\serverName\IST\SQLtxt\AGR). This location centralizes file management and supports version control, backups, and maintenance. IST interpreter will look for all of files in the folder
pathToDataEntry	varchar (255)		Specifies the path to an external data entry program when an alternative interface is required instead of the standard IST-generated entry module.
ValidFrom	datetime	pk	The start date and time from which the application version becomes active. Note: Applications with ValidFrom values for the year 2100 or greater are excluded from production uploads.
ValidTo	datetime		The date and time until which the application remains valid.
IdDeveloper	char (4)		The identifier code of the application's responsible developer.
IdStatistician	char (4)		The identifier code of the statistician responsible for the survey or project.

****Metadata Inheritance*

The _IST table supports metadata inheritance, enabling efficient reuse of existing configurations. This functionality reduces redundancy and ensures consistency when multiple applications share similar structures or logic. Inheritance can be applied selectively to:

- Tables and their field definitions
- Logical control rules
- Report procedures

For example, a supervisory application can inherit the metadata from a field interviewer's application, with additional fields or logic added as needed. This approach simplifies maintenance: changes made in the source application are automatically propagated to dependent applications.

Example:

If we have new application: NewApp (in IST) with next values:

_IST table	
appCode	NewApp
additionalParameters	IST= NewApp; ISTTables=table1FromSomeApp=table1NFromNewApp, table2FromSomeApp=table2NFromNewApp, etc., ISTReportsProcedures=SomeApp2

This means that application NewApp inherits all metadata from app SomeApp and that table table1FromNewApp in NewApp will inherit all metadata (from ISTTables and ISTTablesColums tables) from table table1NFromSomeApp etc.

By default, NewApp will inherit all metadata from IST, ISTables, ISTTablesColums and ISTRulesLogicalControl tables are inherited from SomeApp, but there is exception for *ISTReportsProcedures. NewApp can inherit metadata from SomeApp (default, there is no need to write ISTReportsProcedures=SomeApp) or from different application SomeApp2

Inheriting1 - default: NewApp
additionalParameters: IST=SomeApp

	IST	ISTables	ISTTablesColums	ISTRulesLogicalControl
NewApp inherits all metadata from	SomeApp	SomeApp	SomeApp	SomeApp

Inheriting2: NewApp
additionalParameters: IST=SomeApp;ISTReportsProcedures=SomeApp2

	IST	ISTables	ISTTablesColums	ISTRulesLogicalControl
NewApp inherits all metadata from	SomeApp	SomeApp	SomeApp	SomeApp2

Inheriting3: NewApp
additionalParameters: IST =SomeApp;ISTTables=table1FromSomeApp=table1NFromNewApp;

	IST	ISTables	ISTTablesColums	ISTRulesLogicalControl
NewApp inherits all metadata from	SomeApp	SomeApp	SomeApp	SomeApp

Inheriting4: NewApp
additionalParameters: IST =SomeApp; ISTTables=table1FromSomeApp=table1NFromNewApp; ISTReportsProcedures=SomeApp2

	IST	ISTables	ISTTablesColums	ISTRulesLogicalControl
NewApp inherits all metadata from	SomeApp	SomeApp	SomeApp	SomeApp2

Usage Notes

- Parameters and commands in this table are not case sensitive.

- Parameters can be combined and separated by semicolons (;) or dollar signs ($) where applicable.
- Centralized management of paths and queries supports reliable backups and consistent environments across IST instances.
- Time-based versioning through the ValidFrom and ValidTo fields allows multiple application versions to be maintained and accessed for historical reference or testing.

_ISTDatabaseConnStrings – Database Connection Definitions

The _ISTDatabaseConnStrings table defines the connection details for data storage databases used by IST applications. Each entry in this table provides the connection parameters required for accessing the relational database associated with a specific survey or application.
This table works in close integration with the _IST table through the databaseAlias field and is referenced by the vBazaServerEng view in the IST metadata database. This structure ensures consistent and secure database access across all IST modules.

Purpose
The _ISTDatabaseConnStrings table centralizes the management of connection strings and database aliases, allowing the IST interpreter to:

- Identify the database associated with an application,
- Establish a reliable and secure connection,
- Maintain a unified structure for multiple applications operating within the IST platform.

Field	Type		Description
databaseAlias	nvarchar (10)	pk	A logical name for the data database. This alias is referenced in the _IST table within the databaseAlias field, creating a direct link between the application definition and its underlying database.
databaseType	char (10)		A label indicating the type of database. This field is a legacy element from earlier environments where both DB2 and Microsoft SQL Server were supported. Since IST now operates exclusively on Microsoft SQL Server, this field should always have the value SS.
ODBC	nvarchar (255)		The connection string for establishing a link to the data database. Despite the field name suggesting ODBC, the connection string can also represent OLE DB or other compatible formats. Examples: DATA SOURCE=servername; INITIAL CATALOG=databaseName; Data Source=serverIPaddress; Initial Catalog= databaseName; User Id=XXXX; Password=YYYY; Data Source=(LocalDB)\mssqllocaldb;Initial Catalog= databaseName; Note: all connection strings, as default, are with next parameters: Integrated Security=SSPI;PersISTSecurity Info=False;
OLE_NA	nvarchar (255)	-	Not used
dboowner_NA	char (8)	-	Not used

dboprefix_NA	char (25)	-	Not used
ValidFrom	Datetime	pk	The start date indicating when the database connection becomes valid.
ValidTo	Datetime		The end date indicating when the database connection is no longer valid.

Usage Notes

- The order of parameters in the connection string is not significant.
- Parameters must be separated by a semicolon (;).
- The databaseAlias field serves as the primary link between applications and their corresponding databases.
- Only Microsoft SQL Server is supported in current IST implementations.
- Connection strings must be maintained accurately to avoid access or processing issues.
- Time-based versioning via ValidFrom and ValidTo fields ensures accurate historical tracking and supports updates to database configurations without impacting active applications.

_ISTTables – Tables Structure Definitions

The _ISTTables table defines how datasets are structured within IST applications. It is essential for creating and managing relationships between tables, enabling master-detail (parent-child) setups that support organized, efficient data entry, editing, and processing.

Purpose

The _ISTTables table stores metadata about each table within a statistical application. This metadata enables IST to understand:

- The relationships between tables,
- How data should be organized and linked, and
- How forms and interfaces should dynamically adjust to reflect those relationships.

Master-Detail (Parent-Child) Relationships

IST fully supports master-detail (parent-child) structures, which are commonly required in statistical survey applications.
A master-detail form:

- Displays information from a master table alongside one or more related detail tables,
- Groups rows from the detail table(s) in relation to the corresponding row from the master table,
- Ensures consistency when multiple records are associated with a single master entry.

In relational databases, the master table is linked to one or more detail tables through key fields. Conversely, a detail table can itself act as a master table for another set of linked detail tables, allowing for multi-level hierarchical structures.

Implementation in IST

In IST, these relationships are managed through the typeOfTableParentChild field in the _ISTTables table. This field defines:

- Whether a table is a master table,
- Whether it is a detail table linked to a master table, or
- Whether it exists independently, without parent-child dependencies.

By defining these relationships in the metadata:

- IST dynamically generates forms that reflect the hierarchical structure of the data,
- Developers avoid manually coding relationships, reducing complexity and ensuring consistency,
- Applications can efficiently support complex workflows where multiple detail tables reference a single master record.

Usage Notes

- Correct configuration of master-detail relationships is essential for accurate data collection and editing.
- Each detail table entry must include a master Rowset property, clearly pointing to the master table to which it is linked.
- Proper use of typeOfTableParentChild ensures that applications handle navigation, validation, and updates consistently across related tables.

	Field	Type		Description
1	appCode	char (8)	pk	Code of the application Note: characters /, * should not be used
2	tableName	nvarchar(50)	pk	Data table name Must be same as data table name in relational database, including schema. Schema dbo is default and there is not need to write prefix dbo. Note: characters /, * should not be used
3	typeOfTableParentChild	nvarchar(1)		Table type
4	parentTableName	nvarchar(50)		Parent table name Must be same as data table name in relational database, including schema. Schema dbo is default and there is not need to write prefix dbo. Note: characters /, * should not be used
5	tableDescription	nvarchar(100)		Table description
6	dataEntryScreenOrder	int		Used only if the table is a D (child table) table – specifies the physical location of this table on the generated data entry screen
7	tableOrder	nvarchar (50)		Specifies the order (if more than one table exists) in which tables are displayed in the selection form for data entry or data editing. Default - tables will appear in alphabetical order. Table with table order 0 will not appear in selection form.

8	additionalAlias_NA	nvarchar (10)	-	Not used
9	parentAtributes	nvarchar (2500)		Used for determining the level of editing on the data entry screen and for adjusting the presentation of the data entry table
10	childAtributes	nvarchar (2500)		Used for determining the presentation of the child table (in form of a grid) at data entry
11	ValidFrom	datetime	pk	The start date indicating when the table becomes valid.
12	ValidTo	datetime		The end date indicating when the table is no longer valid.

Detail explanation of fields

Field number 3: typeOfTableParentChild

Represents: Table type
Default value: G
Syntax:

- G – parent table (master table)
- D – child table (detail table)
- A – address book
- K – code book
- S – Support Table
  Indicates that the table is used exclusively by the IST CAPI preparation module to generate CREATE TABLE and INSERT SQL scripts for data transfer.
  Note: Fields for tables marked with S should not be defined in the _ISTTablesColumns table.
  Example: This is useful when a laptop database requires a script for a codebook that is not actively used in the IST application itself, or is only used indirectly, such as for populating a combo box or an auto-complete text box.
- O – Advanced Search Table
  Marks tables that are visible only in the Advanced Search module and not in data entry or data editing interfaces. The Advanced Search functionality relies on the field definitions in _ISTTablesColumns, so for all tables marked with O, the corresponding field metadata in _ISTTablesColumns must be completed.

Field number 5: tableDescription

Represents: System.Windows.Forms.Label on the beginning of the form
Default value: tableName - tableDescription
Parameters:

- fontBold - the title label will be bold
- fontItalic - title label will be italic
- fontUnderline - the title label will be underlined
- $ as first character in the field - title label text is tableDescription text
- lRightA - the title label will be right aligned
- lCenterA - the title label will be centered
- lHeight*N* - height of the title label will be N

- backColorNameOfTheColor - set back color of title label
- foreColorNameOfTheColor - set fore color of title label
- fontSize*N* - font size of the title label text will be N

Usage Notes

- Parameters in this table are not case sensitive.
- The order of parameters in the connection string is not significant.
- Parameters must be separated by a semicolon (;).

Field number 6: dataEntryScreenOrder

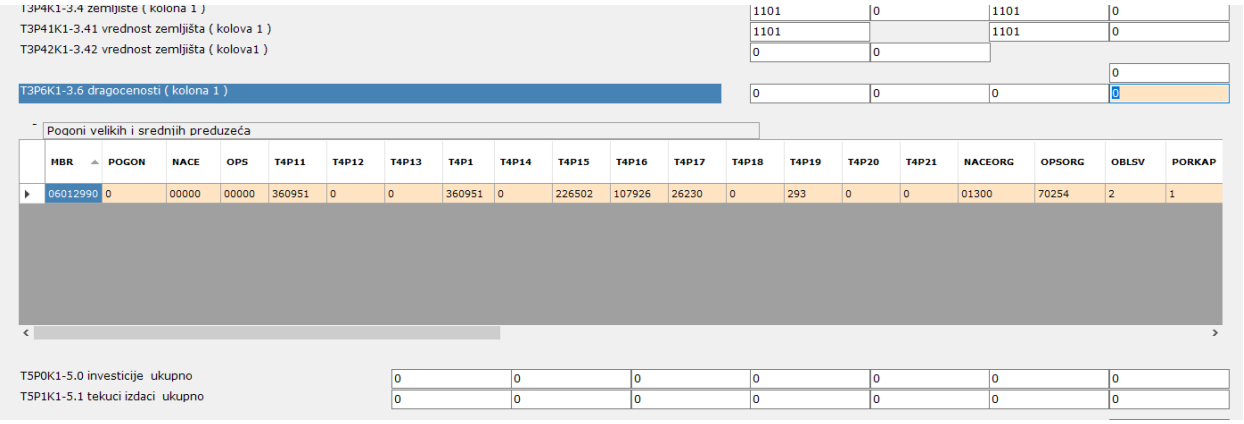
Represents: Physical position of child table on the generated data entry screen
Example:
Partial view of the _ISTTables table (just few columns):

appCode	tableName	typeOfTableParentChild	parentTableName	dataentryScreenOrder
INV01	Podaci_INV01	G		
INV01	Podaci_INV02	D	Podaci_INV01	4

Partial view of the _ISTTablesColumns table (just few columns and rows) from table Podaci_INV01 (main, parent, master table) :

appCode	tableName	columnName	orderNumber	screenPosition
INV01	Podaci_INV01	filed1	30	3.1.1
INV01	Podaci_INV01	filed2	40	3.1.2
INV01	Podaci_INV01	filed3	50	3.1.3
INV01	Podaci_INV01	filed11	60	5.1.1
INV01	Podaci_INV01	filed12	70	5.1.2
INV01	Podaci_INV01	filed13	80	5.1.3

Child table will be position between field3 and filed13 on form for Podaci_INV01 table:



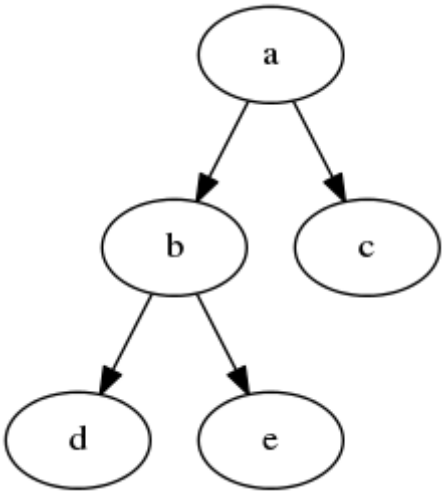
PARENT-CHILD (MASTER-DETAIL) STRUCTURE

Field number 9: parentAtributes Parent-Child (Master-Detail) Structure

IST fully supports Parent-Child (Master-Detail) form structures, enabling complex hierarchical relationships within applications:

- A parent table (e.g., table A) serves as the primary record holder, providing context for related data.
- Child tables (e.g., tables B and C) are linked to the parent table, holding associated detail records.
- A child table (e.g., table B) can simultaneously function as a parent table to another table (e.g., table D), allowing for multi-level relational hierarchies that reflect complex data models.

This flexible structure allows IST applications to manage nested relationships efficiently, ensuring accurate data capture and processing across multiple levels of a statistical workflow.



The parentAttributes field applies to the primary parent table and to any child table acting as a parent in a multi-level relationship. For instance, when Table B serves as the parent of Table D, these attributes are active when the IST form for Table B is accessed.

Field number 10: childAttributes

The childAttributes field in the _ISTTables metadata table defines how child tables are presented in IST forms. It controls child form appearance, behavior, DataGridView design, button events, and panel-based layouts. By default, IST represents a child table with:
- A Button control (showing the child table name/description).
- A DataGridView control to display child records linked to the parent record.
- Default behavior:
DataGridView.Height = 80
Double-click on row header → opens child record form with selected record
Button click → opens empty child form (unless 1–1 relation, then opens child record)
DataGridView columns = all columns from child table
Multiple child tables → buttons arranged vertically.

Usage Notes

- Parameters in this table are not case sensitive.
- The order of parameters in the connection string is not significant.
- Parameters must be separated by a semicolon (;).

_ISTTableColumns – Columns Definition

The fields, defined in this table, will show on the form (data entry, data editing).

Field Requirements and Control Types

- All fields, except virtual fields, must exist in the data table on the MS SQL Server.
- Fields that exist in the database but are not defined in IST metadata will not appear on the form.
- Virtual fields do not exist in the database. Instead, if an expression is defined in the columnAttributes metadata field, the value of that field will be computed dynamically. Controls of this type are referred to as virtual fields.

Control types supported in IST include:

- TextBox
- ComboBox (DropDownList)
- LinkLabel
- CheckBox (including grouped CheckBoxes)
- RadioButton (including grouped RadioButtons)
- DateTimePicker
- Button
- AutoCompleteTextBox
- DataGridView
- VirtualField

Note on virtual fields recalculation:

- By default, a virtual field is recalculated whenever the OnValidating event of any control is triggered.
- The parameter :PK can be used to restrict recalculation so that it occurs only when primary key fields are validated.
- The parameter :CalculateOnExit=FieldName can be used to specify that the virtual field is recalculated only when leaving the listed control(s).

Virtual Field Parameters

Parameter	Description	Usage Example
:PK	Restricts recalculation of the virtual field only to the validation of primary key fields.	#FP{VirtualFieldName}:PK
:CalculateOnExit=Field	Recalculates the virtual field only when leaving the specified control (field).	#FP{VirtualFieldName}:CalculateOnExit=U010101
(default)	Without parameters,	–

Parameter	Description	Usage Example
	the virtual field is recalculated on every OnValidating event triggered by controls on the form.	

	Field	Type		Description
1	appCode	nvarchar (8)	pk	Code of the application Note: characters /, * should not be used
2	tableName	nvarchar (50)	pk	Data table name Must be same as data table name in relational database, including schema. Schema dbo is default and there is not need to write prefix dbo. Note: characters /, * should not be used
3	columnName	nvarchar (100)	pk	Data column name in the table, or the name of a virtual field. For standard fields, the form control name equals the column name
4	primaryKey	nvarchar (1)		Marks if the field is part of the table's primary key (P).
5	relationalDatabase	nvarchar (100)		Logical name of the database (from _ISTDatabaseConnStrings) if the field is linked to another table in database (e.g., for consult datasets).
6	relationalTable	nvarchar (100)		Name of the related (linked) table
7	relationalColumn	nvarchar (255)		Name of the field/column in the related table to which the current field is linked.
8	controlSize	nvarchar (50)		Control size on form (cntl.Width, cntl.Height)
9	validatingEvent	nvarchar (MAX)		Event/action executed on leaving the control (OnValidating event).
10	enterEvent	nvarchar (4000)		Event/action executed on entering the control (OnEnter event).
11	defaultValue	nvarchar (50)		Default value assigned to the control (cntl.DefaultValue).
12	label	nvarchar (max)		Label text displayed with the control.
13	orderNumber	int		Sequence number of the field on the form (cntl.tabIndex).
14	columnType	char (15)		Data type of the field in the database (e.g., char, int).
15	columnLenght	char (10)		Maximum length of the field in the database (cntl.Length).
16	tableLayout	char (20)		Position of the form control, if it belongs to a set of controls displayed in table layout
17	ValidFrom	datetime	pk	Start date from which the column (form control) is valid.

18	ValidTo	datetime		End date until which the column (form control) is valid.
19	columnAtributes	nvarchar (4000)		Additional parameters controlling field behavior, such as expressions for labels, control type, or appearance.

Detail explanation of fields

Field number 3: columnName

- Defines the name of the data column in the table, or the name of a virtual field.
- For both real and virtual fields, the form control name equals this column name.
- Special case: if the name starts with 'nonFilter', the column will not be shown in column selection or filters.

Field number 3: primaryKey

- Marks whether the field is part of the table's primary key.
- Value: P.

Field number 5: relationalDatabase

- Required when the Konsult (Consult) function is used in _ISTRulesLogicalControl.
- Must reside on the same SQL server as the main data database.
- Example: TerritorialDatabase.

Field number 6: relationalTable

- Required when the Konsult (Consult) function is used.
- Specifies the related table within the relational database.
- Example: MunicipalityDataTable from TerritorialDatabase.

Field number 7: relationalColumn

- Required when the Konsult (Consult) function is used.
- Defines the lookup column in the related table.
- On validation of a non-PK field, IST will show a DataGrid with results from:
- SELECT relationalColumn FROM relationalDatabase.prefix.relationalTable
- Selecting a row assigns the value to the control.
- Example: MunicipalityCodeDataColumn from MunicipalityDataTable in TerritorialDatabase.

Field number 8: controlSize

- Defines control width and height on the form.
- Syntax: width;height (integers, 11–999).
- Default: Width=100, height = Windows control default.
- For table layouts: if >9 columns in a row or total width >999px, IST auto-fits the table to the window.

Field number 13: orderNumber

Represents tab order (tabIndex). If using table layout, the effective order combines orderNumber with tableLayout.

Field number 15: columnLenght

Represents maximum text length (TextBox.Length). The value 'MAX' is treated as no limit.

Usage Notes

- Parameters in this table are not case sensitive.
- The order of parameters in the connection string is not significant.
- Parameters must be separated by a semicolon (;).

_ISTRulesDataValidation - Data Validation Rules

The _ISTRulesDataValidation table defines the rules for batch data validation. Batch validation ensures that data remains consistent, accurate, and free of logical errors across records and tables.
In IST, validation rules can be applied at two levels:

- Record level – executed when the cursor leaves a control (onValidation event) or when the user presses the *Data Validation* button on the form.
    - Rules can be defined in two places:
        - The _ISTTablesColumns metadata table (in the validationEvent column).
        - The _ISTRulesDataValidation metadata table.
- Table level – executed through the IST *Batch Data Validation* module.
    - Rules are written exclusively in the _ISTRulesDataValidation table.
    - Validation is applied only to records at the selected time point (ISTYear=YYYY and ISTMonth=MMM).

Table-level data can be validated in two ways:

- Across all rows for the selected time point (on one or multiple tables).
- Across filtered rows matching additional criteria (e.g., ISTYear=YYYY and ISTMonth=MMM and someField=XXX).

Users can choose to run validation against:

- All available rules, or
- A subset of selected rules.

Rules are defined as the WHERE clause of an SQL statement and must be written to detect conditions representing errors.

Execution Flow in the .NET Interpreter

1. The user selects a application and a time point in IST (year and/or month).
2. The .NET interpreter reads the metadata database for these parameters and loads them into RAM.
3. When the user runs the *Batch Data Validation* module, the interpreter generates UPDATE SQL statements in the following form:
   UPDATE tableName
   SET errNumber = 1
   WHERE (ISTYear = YYYY AND ISTMonth = MMM) AND (ErrorCondition)
    - tableName, errNumber, and ErrorCondition are defined in the _ISTRulesDataValidation metadata table.
    - The errNumber column must exist in the corresponding data table.

- ISTYear and ISTMonth are optional but are added automatically if they exist in the table.

4. The prepared SQL statements are executed sequentially. Running them one by one avoids deadlocks and improves MS SQL Server performance.

Important Notes

- Each survey/application must define at least one error rule (globally, not per table).
- If no logical controls are needed, you can still define a placeholder error with a condition such as 2=3.

	Field	Type		Description
1	appCode	nvarchar (8)	pk	Code of the application Note: characters /, * should not be used
2	tableName	nvarchar (25)	pk	Data table name Note: characters /, * should not be used
3	errNumber	varchar (18)	pk	Name of the column in the database table (*tableName*) reserved for error flags. This column must exist in the table and its data type in MS SQL Server must be bit
4	Error	nvarchar (4000)		Defines the WHERE clause of the SQL query. The condition must be written so that it evaluates to true when an error exists
5	condition	nvarchar (255)		Defines when and where (in IST interpreter) the validation rule should be executed
6	errTitle	nvarchar (255)		Text description of the data validation rule or error
7	errAction	nvarchar (255)		The SQL query or program that can be executed through the Automatic Data Correction option. *The SQL query or program must be saved as a text file with the extension .ak. IST interpreter looks for this file in the folder specified by the *pathToQuery* field in the _IST metadata table.
8	errWeight	char (1)		Error weight
9	ValidFrom	datetime	pk	Start date from which the rule is valid.
10	ValidTo	datetime		End date until which the rule is valid.

Detail explanation of fields

Field number 4: error

Defines the WHERE clause of the SQL query. The condition must be written so that it evaluates to true when an error exists (e.g., fieldValue < 500). Supports SQL functions and IST extensions (FP{}, #MOD11{}, #Relacija, etc.).

Field number 5: condition

Defines when and where (in IST interpreter) the validation rule should be executed

- Blank/Empty – Checked in Data Validation, Data Edit, and Data Entry modules
- ExecuteOnlyOnDataEntryForm – Checked on record level (validation event or Data Validation button in Data Entry Form)
- ExecuteOnlyFromDataValidationModule – Checked on table level (batch validation in Data Validation or Data Edit modules)
- ExecuteOnlyOnAddedButtonControl – Checked when a user Button control runs exec RunDataValidationWithPrimaryKey on Data Entry Form
- DontSave – Record with error will not be saved to database
- DontSend – Record with error will not be sent during CAPI synchronization
- Parameters are not case-sensitive; multiple can be combined with ;

Field number 7: errAction

The SQL query or program that can be executed through the Automatic Data Correction option. Must be saved as file with extension ak (file.ak) in the folder specified by *pathToQuery* in the _IST metadata table.

- fileName.ak → executed manually in Automatic Data Correction
- #fileName.ak → executed automatically with batch validation

Field number 8: errWeight

Defines how error highlighting is displayed on the Data Entry Form.
- L – Field value part of error → yellow background
- Blank/T – Field value part of error → red background
- G – Field value part of error → green background
- P – Field value part of error → purple background
- N – Field color not changed. Clicking on an error in the error panel will color all related controls red

_ISTReportsProcedure– Reports, Procedures and Queries

The _ISTReportsProcedures metadata table defines reports generated by executing SQL queries or stored procedures. The logic is fully flexible, supporting any SQL that can run in SQL Server Management Studio (SSMS).

- Report definition – Each report or procedure is defined in the query field. Multiple SQL statements or stored procedures can be included if separated by ;.
- Comments – Any line starting with -- is treated as a comment and ignored during execution.
- Execution flow –
    1. User selects a report/procedure.
    2. .NET interpreter reads the query field into RAM.
    3. Interpreter sets parameter values if required.
    4. Query is executed on MS SQL Server.
- Result handling –
    - Returned result(s) are stored in RAM as one or more data tables in a dataset.
    - Dataset is exported to an Excel file.
    - If a corresponding Excel macro exists, it is executed automatically.

- Headers – The final Excel output is matched with a pre-defined header file (.xml, .xls, .xlsx, .xlsm).
    - For .xls headers, a maximum of 5 rows is supported.
    - For .xlsx, .xlsm, or .xml, the number of header rows is unlimited.
- Dynamic references – Header cells may use IST time variables:
    - GGG or YYY → Year from the IST time point.
    - MMM → Month from the IST time point.

Field	Type		Description
appCode	nvarchar (8)	pk	Code of the application Note: characters /, * should not be used
SeqNumber	int	pk	Sequence number of the report
Title	nvarchar (255)		Name of the report/procedure
visibility	nvarchar (250)		set visibility of report/procedure based on logged in user #{VisibleFalseIf=(select dbo.f_NePrikazuj())}
pivotGraphs	nvarchar (250)	-	.NET interpreter generates a different type of Pivot tables in Excel using builtin macros ISTmPivotChart.xlsm and ISTmPivotChartIndex.xlsm Built in macros are stored in parent directory of pathToQuery\IST
query	nvarchar (255)		SQL query/stored procedure (fileName.sql) put in the field pathToQuery (represents folder path) of the _ISTmetadata table
fileOrLinkForOpening	nvarchar (1000)	-	Any kind of file or link that should be opened. Default program on user's device will be used Example: ZR2019.accdb opens MS Access database methodology.docx opens docx file https://app.powerbi.com/view?r=eyJrIjoiMzIxYzFkMDUtNzE0ZS00ODE2LWE0NzAtMDQzYzE0YmMyYjU2IiwidCI6ImJhZGViOWNiLWU4MjMtNDlmMy1iZWEyLTdiOTlkMjU4ZTA0YiIsImMiOjl9 opens POWERBI report movie.avi opens movie All files should be put in the field pathToQuery Dir (represents folder path) of the

			_ISTmetadata table
messageBeforeQuery	nvarchar (255)		Message that is shown before the execution of the SQL query/stored procedure
macroExcel	nvarchar (255)		Excel file name where the macro is located (macro must have the same name as this excel file). Macro from this file will be executed after copying the query results stored in RAM memory dataset into the Excel table This file is, also, stored in the field pathToQuery (represents folder path) of the _ISTmetadata table
WebMonitoringWithParameters_NA	nvarchar (1000)	-	Not used outside of SORS
ExcelHeader	nvarchar (255)		Reports' headers in various Excel and Excel spreadsheet 2003 XML format
docFormat	nvarchar (50)		used only if report must be some word document
ValidFrom	datetime	pk	Start date from which the report/procedure is valid.
ValidTo	datetime		End date until which the report/procedure is valid.

6.3 Additional Metadata Tables (Multi-Language Support)

_ISTLanguage - Language Settings and Localization

Defines the set of languages supported by IST applications.

Field	Type		Description
IDLang	nchar (10)	pk	ID Language
language	nvarchar (50)		Language

Example values:

- BG – Bulgarian
- EN – English
- ITA – Italian
- MKD – Macedonian
- MNE – Montenegrin
- RUS – Russian
- SHQ – Albanian
- SRC – Serbian Cyrillic
- SRL – Serbian Latin
- UZ – Uzbek

- VN – Vietnamese

_ISTAwls - Interactive Guidance Labels

Defines AlwaysVisibleLabels (AWLs), which are labels tied to form controls. They appear when a user enters a control or presses F1, and disappear when the user leaves the control. In code-behind this is a System.Windows.Forms.Label.

Purpose

Provides additional guidance to users, often used in multi-response questions to display all possible answers or to show detailed control descriptions.

Multi-language support

AWLs can be localized. For this purpose, the _ISTAWLs metadata table stores label definitions in multiple languages.

Field	Type		Description
appCode	nvarchar (8)	pk	App code
tableName	nvarchar (50)	pk	Data table name
columnName	nvarchar (100)	pk	Data column name in data table or virtual field name (for both, form control name is same as data column name)
validFrom	datetime	pk	Initial validity time point
IDLang	nchar (10)	pk	ID Language
IDAwl	smallint	pk	ID AWL
QMT	nvarchar (3)	pk	Possible values: Q represents question M represents possible answers (M1, M2, M3…MF2,MF3, M01,etc2, Letter M will not be shown in AWL label's text) T represents text after all modalities
Sort	int		sort (in order to make order for QMT)
ttext	nvarchar (4000)		text for QMT it is allowed to use virtual fields values Example: <bold>In which province did ' & #FP{fpNAME} & ' work?
validTo	datetime		Final validity time point
onF1	smallint		if has value 1 AWL will be shown only when user presses F1 function key

_ISTLabels - Field and Form Labels

Defines multi-language support for all labels in IST, including:

- Built-in control captions
- Controls' labels
- Custom IST labels

This table ensures that every label displayed in IST applications can be localized into supported languages.

Field	Type		Description
appCode	nvarchar (8)	pk	App code
tableName	nvarchar (50)	pk	Data table name
columnName	nvarchar (100)	pk	Data column name in data table or virtual field name (for both, form control name is same as data column name)
validFrom	datetime	pk	Initial validity time point
IDLang	nchar (10)	pk	ID Language
IDLabel	smallint	pk	ID label
LType	nvarchar (50)		Possible values: Title: td - means table description, in this case columnName has value - for Master table represents title on data entry/edit form for Detail table represents button caption text and title on data entry/edit form Controls: lbl - control's label's text fp- virtual field text (when virtual field represents label) button - Button caption text IST dataentry builtin controls and their value for this field: btnSaveNext btnSave btnDataValidation lblDataValidationOptions btnExit btnNext btnSendEmail gbTimePoint IST datagridview control: dgvTitle - IST control DataGridView's caption (title) dgvColumns - columns of IST control DataGridView dgvAddColumn - text of added columns

			dgvLinkText - links' texts New questions group buttons: buttonNQG - button caption buttonNQGtooltip - button tooltip (columnName has first field in group columnName value)
Ttext	nvarchar (4000)		text in text will be translated in vbCrLf (new line)
validTo	datetime		Final validity time point

_ISTMessages - Application Messages and Notifications

Provides multi-language support for all IST messages.

- Messages are defined at the application level
- They apply to the entire application (not limited to a specific table or field in metadata)

This ensures that system and application messages can be displayed in multiple languages consistently across IST applications.

Field	Type		Description
appCode	nvarchar (8)	pk	App code
validFrom	datetime	pk	Initial validity time point
IDLang	nchar (10)	pk	ID Language
IDMsg	smallint	pk	ID Msg
Ttext	nvarchar (4000)		text in text will be translated in vbCrLf (new line)
validTo	datetime		Final validity time point

_ISTQuestionExplanation - Field-Level Help and Instructions

Supports labels linked to each control that appear when the user enters the control or presses F1, and disappear when the user leaves.

- Implemented as System.Windows.Form.Label in code behind.
- Called QuestionExplanation labels in IST.
- Acts as an addition to AlwaysVisibleLabel.
- Typically used for methodological explanations, hints, or interviewer guidance.
- Fully supports multi-language text.

Field	Type		Description
appCode	nvarchar (8)	pk	App code
tableName	nvarchar (50)	pk	Data table name
columnName	nvarchar	pk	Data column name in data table or virtual field name

	(100)		(for both, form control name is same as data column name)
validFrom	datetime	pk	Initial validity time point
IDLang	nchar (10)	pk	ID Language
IDQuestionExplanation	smallint	pk	ID Question Explanation
Ttext	nvarchar (4000)		text in text will be translated in vbCrLf (new line)
validTo	datetime		Final validity time point
briefdescription	nvarchar (255)		brief description of question, to be used in other IST modules (advanced search, for example)
Questiontext	nvarchar (255)		short question text, to be used in other IST modules (advanced search, for example)

6.4 Reserved Words

Parameters and events in parentattributes field in _isttables (serbian isttabele.t1)

Notation and Conventions

This section describes the shorthand notations and reserved words used throughout the IST Developers' Guide.

d.field – Refers to the current value of the specified field as entered on the form (screen).
Example: d.mbops means the value of field 'mbops' from the current form instance.

ISTQC – Reserved word representing the current Questionnaire Completed (QC) status of the record.
Values: 0 = not valid or not completed; 1 = valid and ready for synchronization.

ISTNote – Refers to the record-level note stored in the ISTNote column of the data table.
Triggered via OnF5=ISTNote event.

LC – Logical Control. Refers to rules defined in the _ISTRulesLogicalControl table, executed either on field exit or via the LC button on the form.

MEM – Alias used in select SQL statements to reference data held in RAM for faster access during validation or calculation.

Parameters are case-insensitive, order-independent, and separated by ';'.

Ist Parameter Syntax Guide

General Syntax

#{ ActionName= TargetGroupsSeparatedBy$; ActionNameIf= ConditionGroupsSeparatedBy$; }

- ActionName: the property or behavior you want to control (e.g., ReadOnly, VisibleFalse).
- TargetGroups: one or more groups of controls, separated by $.
  Inside each group, list multiple controls separated by commas: ctrlA,ctrlB.
- ConditionGroups: one or more conditions, separated by $.
- Mapping rule: the *i*-th target group applies when the *i*-th condition is true.

Separators

- ; → separates parameters inside #{...}
- $ → separates groups (targets or conditions)
- , → separates controls within a group

Example

#{ReadOnly=field1,field11,field112$field3,field33;
ReadOnlyIf=isnull(field8,'2')='2'$isnull(field88,'2')='2';}

- Rule 1: If isnull(field8,'2')='2', set field1, field11, field112 to ReadOnly.
- Rule 2: If isnull(field88,'2')='2', set field3, field33 to ReadOnly.

Usage Notes

- Parameters in this table are not case sensitive.
- The order of parameters in the connection string is not significant.
- Parameters must be separated by a semicolon (;).

Common Action Parameters Reference

Action	Purpose	Typical Use
ReadOnly / ReadOnlyIf	Makes controls non-editable when condition is true.	Lock sensitive fields after entry.
VisibleFalse / VisibleFalseIf	Hides control(s) when condition is true.	Skip irrelevant questions.
VisibleTrue / VisibleTrueIf	Ensure controls are visible when condition is true.	Show fields only for certain conditions.
EnabledFalse / EnabledFalseIf	Disables (grays out) controls when condition is true.	Prevent input until prerequisite is met.
EnabledTrue / EnabledTrueIf	Enables controls when condition is true.	Allow input dynamically.
AssignWhat / AssignTo / AssignIf	Assigns values to other controls (real or virtual).	Auto-fill dependent values.
ColorTo / ColorWhat / ColorIf	Changes control color based on conditions.	Highlight errors or warnings.
SkipTo / SkipIf	Moves focus on another control when condition is true.	Implement skip logic.
StopIf / StopMsg	Stops processing and shows message if condition is true.	Hard validation rules.
DeleteFrom / DeleteIf / DeleteWhere	Deletes records or child rows when conditions are met.	Data cleanup logic.
Exec / ExecMsgIs / ExecMsgIf	Executes stored procedure (optionally with messages).	Advanced data

How the parameter syntax works

IST uses a compact pattern to apply one or more actions to one or more controls, optionally conditioned by one or more expressions.

Core pattern

#{ Action= TargetGroupsSeparatedBy$; ActionIf= ConditionGroupsSeparatedBy$; }

- Actions: e.g., ReadOnly, VisibleFalse, EnabledTrue, AssignTo/AssignWhat, etc.
- Target groups: one or more groups of controls, each group separated by $.
    - Inside a group, list multiple controls with commas: ctrlA,ctrlB
- Condition groups: one or more conditions, also separated by $.
- 1:1 mapping by position: the N-th target group is paired with the N-th condition.

Separators (very important)

- ; separates parameters (key–value pairs) inside #{ … }.
- $ separates groups (targets or conditions).
- , separates multiple controls within a single group.

Matching rules

- Positional pairing:
    - Group 1 of Action= ↔ Group 1 of ActionIf=
    - Group 2 of Action= ↔ Group 2 of ActionIf=
    - …and so on.
- A target group can contain one or many controls (comma-separated).
- A condition group must evaluate to true/false; when true, the action applies to the paired target group.

If you visualize it:

Action= [Group1: ctrlA,ctrlB] $ [Group2: ctrlC] $ [Group3: ctrlD,ctrlE]

ActionIf= [Cond1] $ [Cond2] $ [Cond3]

Pairs: (Group1↔Cond1), (Group2↔Cond2), (Group3↔Cond3)

Tip: Keep the number of $ groups the same on both sides to avoid ambiguity.

Example, explained

#{ReadOnly=field1,field11,field112$field3,field33;

ReadOnlyIf=isnull(field8,'2')='2'$isnull(field88,'2')='2';}

Breakdown

- There is one action: ReadOnly (set listed controls to read-only).
- Targets are split into two groups (separated by $):
    1. field1,field11,field112
    2. field3,field33
- Conditions are likewise split into two groups:
    1. isnull(field8,'2')='2'
    2. isnull(field88,'2')='2'

Pairing (by position)

1. If isnull(field8,'2')='2' is true → set field1, field11, field112 to ReadOnly.
2. If isnull(field88,'2')='2' is true → set field3, field33 to ReadOnly.

So, you've expressed two independent ReadOnly rules in one compact statement.

Quick reference

- Form: #{ Name= targets1$targets2$… ; NameIf= cond1$cond2$… ; }
- Targets: each $ group = one or more controls (comma-separated).

- Conditions: each $ group = one Boolean expression.
- Mapping: group *i* of Name= applies when group *i* of NameIf= is true.
- Separators:
  ; between parameters
  $ between groups
  , within a group

Data Validation

By default, when exiting a field on the form, all rules from _ISTRulesLogicalControl containing that field are executed. Parameters allow limiting which rules to run.

X – Excludes rules that do not contain primary key fields; rules execute only when LC button is pressed.

- If LC is not pressed before saving, validation is performed on save, record is saved, and a message is shown.

A – Excludes rules without primary key fields and without functions Adresar, Konsult, or Relacija.

- Used when validation should ignore these functions.

Note: X and A are mutually exclusive and cannot be combined.

Form Design Parameters

Default behaviors and optional parameters for IST form design:

- Max – Opens the data entry form maximized.
  Syntax: Me.WindowState = FormWindowState.Maximized
- SendEMail – Generates an email button on the form; uses SENDEMAIL folder with Body.txt, Subject.txt and attachments.
  Recipient field is empty unless EMAIL field exists in _ISTTablesColumns.
- ASearch – Forces table to always appear in Advanced Search.
- AlwaysVisibleLabel – Displays a movable, auto-resizing label linked to a control.
  Syntax: AlwaysVisibleLabel(X=130, Y=230)$D='default Text'
  Example: Used to describe field purpose or valid values.
- avlNotMovable – Makes always-visible labels fixed (not movable).
- supportEU – Displays the EU logo on the form.
- buttonLKVisibleFalse – Hides the Batch Logical Control (error checking) button.
- buttonSaveVisibleFalse – Hides the 'Save only' button; 'Save and Next' is renamed to 'Save'.
- gbLKVisibleFalse – Hides the group box for data validation level checkboxes.
- disableMouse – Disables mouse navigation between fields; only Enter key can move focus.
  Behavior: Every field requires Enter to advance.
- shrink – Shrinks fields on screen by removing unused space.
- web – Generates the form for Web (*.aspx, *.aspx.vb).
- MixMode – CATI mode in config + MixMode parameter → IST data entry form used as default.
- CloudCati – Required for CATI data entry through the cloud; generates special connection string.
- DoNotValidateOnArrowUP – Skips validation when moving back with ↑ or Shift+Tab.

- Note – Displays a form-level note (always visible).
  Syntax: Note='someText'$Location(XG=40,YG=290,XGP=40,YGP=30)$Size(1160,245)
- PnlMsg – Uses a custom message control instead of Windows default.
- ScrollBar – Adds a horizontal scrollbar to the form.
- DoNotResize – Prevents autofit of tables; combined with Scrollbar shows scrollbar instead of resizing.
- VerticalScrollVisibleFalse – Hides vertical scrollbar.
- HorizontalScrollVisibleFalse – Hides horizontal scrollbar.
- QCVisibleTrue – Shows 'Questionnaire Completed' checkbox (CAPI only).
- QuestionExplanation – Displays a label when F1 is pressed, providing methodological notes for the field.
- QENotMovable – Makes Question Explanation labels fixed (not movable).
- BackgroundColor – Changes background color of the form.
  Syntax: BackgroundColorRed
- PrimaryKeyDisabled – Disables primary key fields on form load (used for CAPI/CATI).

Events on SaveNext Button

- OnSaveNextDeleteFrom; OnSaveNextDeleteIf – Deletes specified dataTable(s) from RAM when SaveNext is pressed if conditions are met.
  Syntax: OnSaveNextDeleteFrom=subtable1, subtable2; OnSaveNextDeleteIf=condition
  Example: OnSaveNextDeleteFrom=Trace1, Lica; OnSaveNextDeleteIf=U010110='2'
- OnSaveNextFocusOn; OnSaveNextFocusIf – Moves focus to a specific PK field after SaveNext if conditions are met.
  Note: Valid only with OnSaveNextDontClose.
- OnSaveNextDontClose – For child tables only; prevents the form from closing after SaveNext.
  Note: Cancels OnSaveNextCloseIf.
- OnSaveNextCloseIf – Closes or resets the form depending on conditions after SaveNext.
- OnSaveNextSimpleMsgIs; OnSaveNextMsgIf – Displays simple OK-only message(s) after SaveNext if conditions are met.
- OnSaveNextMsgIs; OnSaveNextMsgIf – Displays OK/Cancel message(s) after SaveNext if conditions are met. Focus on Cancel button.
- OnSaveNextNoteIs; OnSaveNextNoteIf – Sets text for form-level NOTE label on SaveNext if conditions are met.
- OnSaveNextStopIf; OnSaveNextStopMsg; OnSaveNextStopMsgFocusOn – Stops execution on SaveNext if conditions are met and displays message.
- OnSaveNextStopMsgOnPnl – Displays a custom message panel and stops execution on SaveNext if conditions are met.
- OnSaveNextStopLoopFromGridIf – Stops automatic looping from DataGridView when SaveNext is pressed and condition met.
- OnSaveNextExec – Executes a stored procedure when SaveNext is pressed.
  Example: OnSaveNextExec=EXEC usp_upd_Kontrola_teren d.god,d.mbr

Events on Exit Button

- OnExitStopIf; OnExitStopMsgIs; OnExitStopMsgFocusOn – Stops execution when Exit button pressed if conditions met.
- OnExitStopMsgOnPnl – Displays custom message panel on Exit if conditions met.
- OnExitSimpleMsgIs; OnExitMsgIf – Displays simple OK-only message(s) on Exit if conditions are met.
- OnExitMsgIs; OnExitMsgIf – Displays OK/Cancel message(s) on Exit if conditions are met. Focus on Cancel.

Events on F5 Key

- OnF5=ISTNote – Displays ISTNote panel when F5 key is pressed and released.
  Behavior: Shows record-level notes stored in ISTNote column (nvarchar, developer-defined length).

QC (Questionnaire Completed) Events

- AfterUpdateQCOn0IstMsg – Displays a message from ISTMessages table after QC updated to 0.
- AfterUpdateQCOn1IstMsg – Displays a message from ISTMessages table after QC updated to 1.
- AfterUpdateQCExec – Executes a stored procedure after QC updated (0 or 1).
  Syntax: AfterUpdateQCExec=Exec usp_xxx param1, param2, ISTQC
  Example: AfterUpdateQCExec=Exec usp_SkloniDatumZakazivanja d.mbops, ISTQC

Return from Child Form event

- OnReturnVisibleFalse; OnReturnVisibleFalseIf – Hides listed controls if conditions are met.
- OnReturnVisibleTrue; OnReturnVisibleTrueIf – Shows listed controls if conditions are met.
- OnReturnEnabledFalse; OnReturnEnabledFalseIf – Disables listed controls if conditions are met.
- OnReturnEnabledTrue; OnReturnEnabledTrueIf – Enables listed controls if conditions are met.
- OnReturnFocusOn; OnReturnFocusIf – Sets focus on listed control if conditions are met.
- OnReturnAssignTo; OnReturnAssignWhat; OnReturnAssignIf – Assigns values to parent form fields if conditions are met.
- OnReturnSimpleMsgIs; OnReturnMsgIf – Displays simple OK-only message(s) when returning if conditions are met.
- OnReturnMsgIs; OnReturnMsgIf – Displays OK/Cancel message(s) when returning if conditions are met. Focus on Cancel.

Question Group Buttons

- NewQuestionGroupButtonLocation – Sets X, Y position of question group buttons.
  Syntax: NewQuestionGroupButtonLocation(x=300, y=65)
- NewQuestionGroupButtonVertical – Displays group buttons vertically instead of horizontally.
- NewQuestionGroupButtonVisibleFalse – Hides question group buttons.

Parameters and events in childattributes field in _isttables (serbian isttabele.t2)

DataGridView Design

- Min – Sets DataGridView.Height = 26.
- Max – Sets DataGridView.Height = 160.

- NoGrid – DataGridView not visible, space reserved; toggle button (+/–) shows/hides grid.
- NoGridSpace – No space reserved for DataGridView.
- NoPlus – Hides the +/– toggle button.
- NoButton – Hides the child table button.
- buttonWidth=N – Sets button width.
- buttonHeight=N – Sets button height.
- forecolorColorName – Sets button foreground color.
- backcolorColorName – Sets button background color.
- buttonLocation(X=xx,Y=yy) – Sets button coordinates.
- buttonCaption=Text – Sets custom button caption text.
- buttonVisibleFalse – Hides the child table button.
- horizontal – Arranges multiple child table buttons horizontally.
- fontSizeN – Sets caption font size.
- fontBold – Makes caption text bold.
- fontItalic – Makes caption text italic.
- fontUnderline – Underlines caption text.
- lRightA – Aligns caption text to the right.
- lCenterA – Centers caption text.
- view – Disables double-click on row header and button click.
- gridColumnsIST – Uses only columns defined in _ISTTablesColumns.
- gridColAutoResize – Auto-resizes all columns to fit contents.
- gridHeight=N – Sets DataGridView.Height = N.

Example: NoGrid;NoPlus;NoGridSpace;buttonWidth=150

Events on Child Table Button Click

- StopIf; StopMsg; StopMsgFocusOn – Stops execution when child button is clicked if condition is met; displays message and sets focus.
- StopMsgOnPnl – Displays custom (non-Windows) message panel when condition is met.

Panel Design

- Panel – Displays all child fields in a panel with Save button + DataGridView.
- Scrollbar – Adds horizontal scroll bar; disables auto-resize.
- disableMouse – Disables mouse navigation on panel; only Enter advances.
- VerticalScrollVisibleFalse – Hides vertical scrollbar.
- HorizontalScrollVisibleFalse – Hides horizontal scrollbar.
- FontSizeN – Sets font size for all panel controls.
- AutoListBy; AutoListTo – Auto-fills panel with next record after Save until AutoListTo condition is reached.
  Syntax: AutoListBy=FieldName; AutoListTo=VirtualFieldValue
  Example: AutoListBy=RBR; AutoListTo=#FP{fprbr};

Events on Save Panel Button

- SkipTo / SkipIf – Defines navigation after saving panel record.

Syntax: SkipTo=panelChildTableName OR panelChildTableName.field OR buttonNEXT; SkipIf=condition
Example: SkipTo=panelKRAJ.U020101; SkipIf=rbr=#FP{fprbr};

- AddRowsTo / AddRowsIf – Adds rows to another child table after saving, if conditions are met.
  Syntax: addRowsToChildTableX(field1, field2,...) values(sourceField1, sourceField2,...); addRowsIf=condition
  Example: addRowsToTRACE2(god,rbr,ime) values(god,rbr,P010100); addRowsIf=T510110 IN ('1','2','3');

Event on Panel Got Focus

- OnEnterSkipTo / OnEnterSkipIf – Skips to specified field(s) when panel receives focus.
  Syntax: OnEnterSkipTo=fieldName; OnEnterSkipIf=condition;
  Example: OnEnterSkipTo=U020101; OnEnterSkipIf=U010110 IN ('1','3','5');

Event on Leaving Last Primary Key Field in Panel

- NoNewRow – Disables adding new rows; only editing of existing rows allowed.

Parameters and events in validatingevent field in _ isttablescolumns (serbian istpolja.od)

Validation and Events

validatingEvent defines behavior when a control loses focus.
Covers: KeyPress, DoubleClick, and Validating events.

KeyPress Event

Special keys supported: F2, F3 (missing value codes), F6 (convert agricultural units to hectares). F2 is refusal and F3 is don't know
F6 - Recalculating agricultural unit of measures to hectares *only for Serbia

- addAgrCalc - value in the textBox will be calculated to Ha (hectare) when next text is entered after digits and F6 function key is pressed
  Example:
  Textbox value is 101.1 ju when user pressed F6 value will be 58.638 (=101.1*0.58)
  ju (jutra) - textBox value=textBox value*0.58
  la (lanac) - textBox value=textBox value*0.72
  ko (kosa) - textBox value=textBox value*0.25
  li (livada) - textBox value=textBox value*0.25
  hv (hvat) - textBox value=textBox value*0.036
  dn (dunum) - textBox value=textBox value0.1
  si (šinir) - textBox value=textBox value*0.1
  dl (dulum) - textBox value=textBox value*0.16
  mo (motika) - textBox value=textBox value*0.07
  pl (plug) - textBox value=textBox value*0.4
  ce (čerek) - textBox value=textBox value*0.045

DoubleClick Event

When double click on TextBox opens file dialog, and path of selected file (with file) passes to Textbox

Validating Event

IST processes type checks, field validation, rules from _ISTRulesLogicalControl, optional pop-ups for Adresar / Konsult / Relacija, and skips, stops, assignments, or formatting.

Simple Data Validation Parameters

Warning – light validation with message and continue;
Min/Max – numeric range;
DecPart – decimal precision;
Yes/No – valid/invalid sets including reserved words (numeric, integer, alpha, alphaNumeric, multiResponseNumeric, multiResponseAlpha, MnResponseCount, MxResponseCount, date, istDate, time);
RequiredIf – conditional requirement;
MnLength/MxLength – length constraints.

Messages

MsgIs/MsgIf – Show OK/Cancel messages if conditions met.
OptionMsgIs/OptionMsgIf/OptionMsgSkipTo/OptionMsgAction – Show option dialogs with custom button text.

Impact on Other Controls

ReadOnly/ReadOnlyIf – make controls read-only.
ColorTo/ColorWhat/ColorIf – apply conditional formatting.
VisibleFalse/VisibleTrue – control visibility.
EnabledFalse/EnabledTrue – control enabled state.

Assignments

AssignWhat – defines value to assign (from control values or AWLValue).
AssignTo – target controls.
AssignIf – condition for assignment.

Stops and Skips

StopIf/StopMsg – stop execution and show message.
SkipTo/SkipIf – skip to another control.
SkippedEnabledFalse, SkippedVisibleFalse, SkippedSetEmpty, SkippedDontSave – define skip behavior.
SkipAction – combine skip behaviors. Common values: -SetEmpty, EnabledFalse, DonSave, VisibleFalse
PageDownTo – unconditional skip to next control.

Deletes and Exec

DeleteFrom/DeleteWhere/DeleteIf – delete records under conditions.
exec storedProcedureName – run stored procedure with optional parameters.
exec RunDataValidationWithPrimaryKey – built-in IST validation.
execMsgIs/execSimpleMsgIs; execMsgIf – messages after execution.
execDialogIs; execDialogIf – dialogs before execution.

CheckBox Group Events

UncheckedAllOtherIfTrue – ensures only one checkbox remains selected in multi-response groups.

Processing Flow Notes

pimary key validation:

record existence check, insert/update logic, auto-assignments.

non-primary key validation:

data type checks, rules, cascading ComboBox updates, virtual field recalculation, skips, assignments, formatting.

Events in enterevent field in _ isttablescolumns (serbian istpolja.do)

Defines behavior executed on a control's OnEnter event (applies to all controls except LinkLabel).

Syntax: #{parametersSeparatedBy;}

Order of events

- Set background color to Bisque (where applicable).
- If the control has a value, select all text.
- Populate AutoCompleteTextBox data source.
- If ScrollDown is listed, scroll the form so the control is first on screen.
- Populate AlwaysVisibleLabel and QuestionExplanation from RAM.
- AlwaysVisibleLabel is shown by default.
- Evaluate Assign rules; set values when conditions are met.
- Apply VisibleFalse/EnabledFalse; then VisibleTrue/EnabledTrue; then ReadOnly flags when conditions are met.
- TextBox only: if convertCirLat/convertLatCir is set, convert characters on TextChanged.

TextBox OnChange Shortcut

- convertCirLat / convertLatCir – converts input to Cyrillic/Latin on TextChanged.

Value assignment (shared syntax)

- AssignWhat= valueFromControlsOrAWLValue $-separated
- AssignTo= control1, control2 $-separated groups
- AssignIf= condition1$condition2

Impact on Other Controls (shared syntax)

- ReadOnly / ReadOnlyIf
- visibleFalse / visibleFalseIf
- visibleTrue / visibleTrueIf
- enabledFalse / enabledFalseIf
- enabledTrue / enabledTrueIf

Examples:

- #{AssignWhat=fp012;AssignTo=p012;AssignIf=isnull(p012,'')='' or p012='0'}
- #{AssignWhat=nonFilterAdminStuff;AssignTo=AdminStuff;AssignIf=2=2;}

- #{ReadOnly=field1,field11,field112$field3,field33;ReadOnlyIf=isnull(field8,'2')='2'$isnull(field88,'2')='2';}

Parameters in defaultvalue field in _ isttablescolumns (serbian istpolja.dg)

DefaultValue

Applies to TextBox, AutoCompleteTextBox, and ComboBox. Sets the control's default value.

- Text – literal default value.
- {P} – keep key value on new record.
- {+} – increment key by 1 on new record.
- {+FPP} – increment key by 1 and focus the first non-key field when the full key is filled.

Parameters in label field in _ isttablescolumns (serbian istpolja.opis)

Label

Default control label. Rendered as System.Windows.Forms.Label with text: controlName – labelText (left-aligned by default).

Syntax (supports multi-row table headers and merged cells): parametersSeparatedBy$Char Text # headerRow1 # headerRow2 ...

parametersSeparatedBy**$**CharText**#**parametersForColumnsHeaderTextSeparatedBy**$**CharColumnsHeaderText(ifAny)SeparatedBy**;**Char**#**parametersForColumnsHeaderTextSeparatedBy**$**CharColumnsHeaderText(ifAny)SeparatedBy**;**Char(NextRow)...

Common parameters:

- multiLine – doubles label height.
- fontBold / fontItalic / fontUnderline / fontSizeN.
- foreColorColorName / backColorColorName.
- $ – omit controlName from label text.
- lRightA / lCenterA – label alignment.
- fRightA – right-align the text within the control.
- # – start header definition; header text centered by default.
- lHeightN – header row height N.

Examples:

- $fontSize8$multiLine$fontBold$textOfTheLabel
- # ;merge;merge;merge;someText | #LHeight2$Total;qty1;qty2;qty3;qty4 | #1;2;3;4;5;
- #merge;merge;merge;merge;TotalNumberOfComputers; ; #lHeight2$; ;merge;merge;in school; ; # ; ; ;merge;with WIFI;

Parameters in columntype field in _ isttablescolumns (serbian istpolja.tip)

ColumnType

Data type validation applied on Validating event. If mismatch, IST shows a message and keeps focus on the control.

Supported types:

- INT / INTEGER / SMALLINT / TINYINT
- BIG / BIGINT
- FLOAT / REAL / DECIMAL / MONEY / SMALLMONEY / NUMERIC (*comma is converted to dot before checking)
- BOOLEAN / BIT
- TIME / DATE / DATETIME / ISTDATE
- CHAR / STRING / TEXT (**leading zeros preserved when {+} is used for PK)
- EMAIL (name@example.com format)

Parameters in tablelayout field in _ isttablescolumns (serbian istpolja.forma)

TableLayout

Aligns controls in a grid on the form (rows/columns). Applies to all controls except DataGridView.

Syntax: table.row.column

Notes:

- Each cell holds one control; controls can span by sizing (longer lengths).
- Use to create compact, consistent layouts across forms.

Parameters in columnattributes field in _ isttablescolumns (serbian istpolja.izraz)

ColumnAttributes

Defines the control type and its properties. Reserved words are case-insensitive, order-independent, and separated by '$'.

TextBox

Default control type if no explicit type is listed.

Parameters

- Font: FontBold, FontItalic, FontUnderline, FontSize*N*
- Text alignment: fRightA, fCenterA
- Border: BorderFixed, BorderNone
- Color: foreColorColor*Name*, backColorColor*Name*
- Behavior: Encrypt, NoTabStop, MultiResponse, MultiLine, ReadOnly, EnabledFalse, VisibleFalse
- DataValidationFor – Marks a column as available for filtering in the Data Validation module. This means that data validation (e.g., batch logical control) can be applied only to the subset of records filtered by the values in this column (for example, validating records that belong only to a single municipality)

AutoCompleteTextBox

Custom IST control (TextBox + ListBox on a Panel). Faster than ComboBox; supports cascading filters.

Key syntax:

- autoComplete=filenameOrDataSource
- autoCompleteLimitToList=True/False (default True)
- autoCompleteStart=Number, autoCompleteLength=Number (for file source)

- autoCompleteMinTypeLength=Number (default 2)
- autoCompleteFilter=d.Control / #FP{VirtualField} (+ Start/Length for file)
- autoCompleteAssignTo=Control (+ Start/Length for file)
- autoCompleteLetter=LAT/CIR
- autoCompleteHeight/Width/MaxDropDownItems
- autoCompleteDataSource=SELECT ...
- autoCompleteDisplayMember=column, autoCompleteValueMember=column
- autoCompleteOrderBy=col1,col2

Common properties (same as TextBox): Font*, fRightA/fCenterA, Border*, foreColor/backColor, Encrypt, NoTabStop, MultiLine, ReadOnly, EnabledFalse, VisibleFalse, DataValidationFor.

ComboBox

Windows ComboBox. Filtered ComboBoxes are supported for PK fields.

- Defaults: DropDown style; Visible/Enabled; Font Regular
- Data: DataSource, DisplayMember, ValueMember, DataFilter
- Question groups: newQuestionGroup
- Other: Encrypt, NoTabStop, EnabledFalse, VisibleFalse, WebStr / endOfWebStr, divDisplayNone / endOfdivDisplayNone, DataValidationFor

Web syntax (CBO/CBW renderers; same syntax):

CBO{ DataSource=SELECT ...; Filter=col=d.Control OR #FP{vf}; DisplayMember=col; ValueMember=col;}

LinkLabel

Hyperlink label. Implemented as a read-only TextBox. Supports newQuestionGroup.

CheckBox

Binary flag (writes 1/0). For multi-response legacy, writes 1..n by order checked.

- Question group: newQuestionGroup
- Other: NoTabStop
- Syntax: CheckBox / CheckBoxL / CheckBoxR (standalone)
- Grouped multi-response: CheckBoxL_GroupName / CheckBoxR_GroupName

RadioButton

Single choice within a group; writes integer value.

- Question group: newQuestionGroup
- Alignment: RadioButtonH / RadioButtonV
- Other: NoTabStop

Syntax examples:

- RadioButtonH{'One','Two','Three'} → values 1…3
- RadioButtonH{0'None',1'One',2'Two'} → writes listed values

- Use 'left' prefix to place labels left of buttons

Button

Triggers EXEC logic in validatingEvent. No backing data column. Label text is taken from 'label' field.

- Defaults: Visible/Enabled; TextAlign=MiddleLeft
- Size: Width=N, Height=N
- Font: FontBold, FontItalic, FontUnderline, FontSizeN
- Color: foreColorColorName, backColorColorName
- Text alignment: lRightA, lCenterA
- Location: ButtonLocation(X=xx,Y=yy)
- Other: NoTabStop
- Question group: newQuestionGroup

DateTimePicker

Selects date/time; data type must be DATETIME. Supports newQuestionGroup and NoTabStop.

DataGridView

GroupBox + DataGridView; display-only by default; no backing data column.

- Defaults: Dock=Fill, ReadOnly=True, Cursor=Hand, MultiSelect=False, no add/delete rows, resizing enabled, row headers visible, clipboard disabled, EditOnKeystroke, TabStop=False, Verdana 8, Border=None

Data properties:

- GridSource=metadataTableName
- GridSourceView=viewName
- GridSourceWhere=where ...
- GridOrderBy=order by ...
- GridColumns=HeaderText=col1, col2, ...
- GridColumnsWidth=w1, w2, ...

Command columns:

- GridLinkText=txt1$txt2 ...
- GridLinkAction=Edit/Delete=tbl/Open=tbl/AddNewRowInGrid/EditRowInGrid/SaveRowInGrid/CancelEditRowInGrid/Exec/Action#Exec
- GridLinkWidth=w1$w2 ...
- GridLinkStopIf=cond1$cond2 ...
- GridLinkStopIfMsg=msg1$msg2 ...
- GridAddColumn=Header=MemTable.Col$...
- GridAddColumnOnPosition=pos1$pos2 ...
- GridColumnLink=col1$col2 ... (make specific cells links)
- GridColumnLinkAction=actions matching GridColumnLink
- EditRowInGridColumns=editableCol1, editableCol2,...

Layout:

- GridHeight, GridWidth, GridRowHeadersWidth, GridColumnHeadersHeight, GridRowsHeight, GridColumnsAutoResize
- GridTitle=Text (+ ForeColor, FontBold/Italic/Underline, FontSize)
- GridLocation(X=, Y=), GridFontSize, GridHeaderRowFontSize
- Flags: NoTabStop, EnabledFalse, VisibleFalse

VirtualField

Rendered as a disabled, non-tab-stop TextBox. No backing database column. Displays constants or computed values.

Label text shortcut:

- 'Text' – treated as static label text; hidden from selection/filter lists in Data Editing.

Expression elements:

- d.Control – value from screen control
- #FP{vf} – value of a virtual field
- Head.Control or {Parent.Control} – value from parent table
- {ChildTable.Control} – value from a child table

CASE/WHEN:

- #{case when cond1 then cmd1 when cond2 then cmd2 else cmdN end}

Select SQL expressions (with modifiers):

- #{select …} – base form
- :F – hide on Data Entry (show on Data Editing)
- :S – show on Data Entry only
- :PK – recalc on PK validation only
- :CalculateOnExit=Control – recalc only when leaving the listed control

RAM and aliases within select:

- D.Control – refers to form's data table
- MEM – use RAM memory DataTable instead of SQL table (e.g., #{select sum(col) FROM child MEM where …})

Time and user parameters:

- MMM (month), GGG or YYY (year), {YYY-x} or {MMM-x} or {YYY+x} or {MMM+x}
- {CAPIUSER}, {CATIUSER}, {APPCODE}

Built-in previous-period helpers (require GOD/MES or ISTYear/ISTMonth):

- PM/PMonth – previous month value for PK
- PG/PY – previous year value for PK

- PK/PQ – previous quarter value for PK
- PP/PH – previous half-year value for PK
  Syntax: #{PM.columnName}

Supported SQL helper functions within expressions: Left, Right, LTrim, RTrim, IsDate, IsNumeric, CharIndex, InStr, Len, Floor, Round, Year, Month, Day.

Functions a in error field in _istrulesdatavalidation (serbian istlk.greska)

These functions can be used inside the error column (WHERE part of the SQL query) to extend validation logic.

Relation function

- #Relacija {SomeTable or SQL; p1,p2,...,pn: assignments; [alwaysfromcodebook]} – Checks if a record exists in a reference table or SQL result.
  SomeTable = lookup table or SELECT query can be used
  p1...pN = key fields to match
  assignments = map columns from lookup to form fields (e.g., code=codeValue: title=titleValue)
  alwaysfromcodebook = (optional) forces fields to always come from lookup, not existing DB record

Validation logic

Batch Validation: Marks records as errors if they don't exist in lookup.
Data Entry Form: Checks lookup in RAM; if no match, shows DataGridView to select valid record. Assignments are executed if record exists.
Supports constants and SQL expressions ({GGG}, {MMM}, {GGG-1}, etc.).

Example:
#Relacija{ (SELECT * FROM RPJ.dbo.f_RPJ_Evs({GGG},{MMM})); opsR=mbops: title3=titleMun, title2=titleMunCir} → fills title3 and title2 on form if matching record found.

#WebRelacija – Same as #Relacija but explicitly used for web collection. This function is used to build selection lists in web forms and allows multiple Relacija definitions for the same table (Relacija, WebRelacija etc.).

Other functions

- isNumeric(fieldName) – Checks if the value is numeric.
  isNumeric(field)=0 → generates error (value contains non-digit).
- #Konsult(fieldName) – Validates if the field's value exists in a related lookup table.
  Query: SELECT relationalColumn FROM relationalDatabase.prefix.relationalTable
  relationalDatabase, relationalTable, and relationalColumn are defined in _ISTTablesColumns.
- #Adresar{table; p1,p2,...,pn} – Validates existence of a record in an AddressBook table.
  table = AddressBook table defined in _ISTTables (type A).
  p1...pN= primary key fields to match.

Constants allowed (single quotes, e.g. '12').
Data Validation Module: Finds missing records in AddressBook.
Data Entry Module: If record does not exist in RAM, a DataGridView with AddressBook values is displayed for selection.

- #MOD11(field, length) – Validates value using modulo-11 checksum.
  length optional: checks both modulo-11 and field length.
- #MBRP(field) – Validates registration number of enterprise.
  Must be 8 characters, modulo-11 check.
- #JMBG(field) – Validates citizen ID (unique personal number).
  Must be 13 characters, modulo-11 check.
- #VOPS(field) – Validates municipality code.
  Must be 5 characters, modulo-11 check.
- #VNAS(field) – Validates settlement code.
  Must be 6 characters, modulo-11 check.
- #VPIB(field) – Validates VAT number.

Parameters in .sql file in query field in _ isttablescolumns (serbian isttabs.upitzatabelu)

Parameters are optional and case-insensitive.

Exclude

--EXCLUDE n – Exclude last n columns from the query result. Works per sheet or for all sheets.

Sheets

- --firstSheet [nameOfTheSheet HeaderParam1='text1', ...] – Defines the first sheet in Excel. Allows naming the sheet and inserting up to 9 header parameter values. In XML spreadsheet 2003 format, header parameters can be referenced in header cells with =HeaderParam*N*

Syntax:
--firstSheet 'nameOfTheSheet' HEADERPARAM1='param1', HEADERPARAM2='param2'
Example:
--firstSheet 'MunicipalityReport' HEADERPARAM1='Belgrade'
Excel header will include parameter value (Belgrade).

- --newSheet [nameOfTheSheet HeaderParam1='text1', ...] Define next sheet(s) with same options as firstSheet

Syntax:
--newSheet 'nameOfTheSheet2' HEADERPARAM1='param1'
Example:
-- newSheet 'MunicipalityReport' HEADERPARAM1='Belgrade'
Excel header, in new sheet, will include parameter value (Belgrade).

Example:
select * from Population;
select * from Households;

Runs two queries sequentially; results exported to one sheets.

--firstSheet 'Population'
select * from Population;
--newSheet 'Households'
select * from Households;
Runs two queries sequentially; results exported to two sheets.

Parameters

--parameter arrayOfParameters – Declares parameters to be entered by the user before executing the SQL/stored procedure.

- Syntax: P1:Label_text P2:Second_parameter ...
- Label text is separated from parameter name with :
- Use _ for spaces in labels (replaced by blanks on the parameter form)
- Parameters can appear as TextBox, ComboBox, or AutoCompleteTextBox (same syntax as _ISTTablesColumns)
- On report execution, a form with input fields is shown; entered values are passed to SQL query.

Example 1:
--parameter P1:Municipality P2:Settlement
select * from data where municipality=@P1 and settlement=@P2
Example 2:
--parameter P1CBO{DATASOURCE = select * from municipalityCodeBook;
DISPLAYMEMBER=municipalityName;
VALUEMEMBER=municipalityCode}:Please_select_municipality P2CBO{DATASOURCE = select * from settlementCodeBook; FILTER=municipalityCode=d.P1; DISPLAYMEMBER=settlementName;
VALUEMEMBER=settlementCode}:Please_select_settlement P3{MnLength=5;
YES=numeric}:Select_ownership
P4:Select_legal_form

Time points

- {YYY}, {MMM} → Replaced with IST year and month at runtime (preferred).
- {YYY±x}, {MMM±x} → Shifted year/month (preferred).

Additional formatting via XML Spreadsheet 2003 format (*.xml)

Header file must be in XML Spreadsheet 2003 format (*.xml). All formatting from the header XML file is preserved in the report.
Example: if column B in the header is bold, report column B will also be bold.

Additional formatting via ISTASA column (ISTASA Formatting)

For additional formatting of rows/columns you can add an additional column with ISTASA alias in the query.

ISTASA column contains a list parameter for formatting.
Order of parameters is not important. It is important to write an ISTASA column before columns that you want to exclude from results.

- Add an extra column with alias ISTASA in the query.
- The ISTASA column contains formatting parameters separated by ';'.
- Order of parameters is not important.
- ISTASA must be placed before excluded columns in the query.

Parameters usable in ISTASA column:

- B – Bold entire row
- I – Italic entire row
- Vn – Set row height to n
- FSn – Set font size of row to n
- R[x] – Replace all field values in the row with x
- Rn[x] – Replace values of column n in the row with x
- C – Center row
- KnB – Bold column n in the row
- KnI – Italic column n in the row
- Kn[x] – Replace values of column n with x
- KnU – Uppercase column n in the row
- KnNn – Indent column n by n levels (1 level = 5 spaces)

Example 1

```
--EXCLUDE 2
select name, age, income, address, code from persons
```

Excludes last 2 columns → address, code won't appear in Excel.

Example 2

```
--EXCLUDE 2
select settlementName as nameTer, sum(Number) as Number,
    sum(numberp120) as numberp120, settlement,
    nuts0+nuts1+nuts2+nuts3+seq1+sequence+settlement as sequence,
    'K1N6' as ISTASA, territory, sequenceOrder
```

Result:

- First column indented 6 levels (30 spaces).
- territory and sequenceOrder excluded due to --EXCLUDE 2.

Example 3

```
select name, value, 'B;K1N4' as ISTASA from table
```

Rows are bold, first column indented 4 times.

Example 4

```
--EXCLUDE 3
select munName as nameTer, sum(Number) as Number,
    sum(numberp120) as numberp120, munCode,
    nuts0+nuts1+nuts2+nuts3+seq1+seq as sequence,
    case when munCode<>'70114'
       then 'B;K1N4'
       else 'B;K1N4;R[...]' end as ISTASA,
    teritory, sequenceOrder, tsO
```

Result:

- Rows with munCode <> '70114': bold + column 1 indented 4 levels (20 spaces).
- Rows with munCode = '70114': bold + replace all values with [...].
- territory, sequenceOrder, tsO excluded due to --EXCLUDE 3.

Example 5

```
select N'unknown municipality' as terName, N'subtitle' as column2,
    sum(number) as number, sum(columnp120) as columnp120,
    nuts1,nuts0+nuts1+'00' as seq,
    'B;K1N1;V20;K2U' as ISTASA
```

Result:

- All rows bold
- Column 1 indented 1 level (5 spaces)
- Row height set to 20
- Column 2 values uppercased

Built-in macro

After results are written into Excel, IST automatically runs ISTMacro.xlms, located in pathToQuery\IST parent directory.

Parameters in .sql file in ltype field in _ isttlabels (serbian istlabels. Tip)

Label Types (LType values)

- td – Table description.
    - For Master table → form title.
    - For Detail table → button caption and form title.

  Note: ColumnName should have value -
- lbl – Control's label text.
- fp – Virtual field label text.
- button – Button caption text.
- ISTLBLxxx – Conditional labels resolved dynamically (via case when ... then ... expressions).

Example:

```
fontBold$foreColorBlack$backColorIvory$
#{case when isnull(d.LiceJe,0)=1 then ISTLBLOL2 when isnull(d.LiceJe,0)=2 then ISTLBLPl2 else '' end}
```

Here, ISTLBLOL2 and ISTLBLPl2 are resolved from _ISTLabels for the given appCode, tableName, and IDLang.

Built-in IST Controls and their values for LType field

- btnSaveNext – Save & Next button caption.
- btnSave – Save button caption.
- btnDataValidation – Data Validation button caption.
- lblDataValidationOptions – Label for validation options.
- btnExit – Exit button caption.
- btnNext – Next button caption.
- btnSendEmail – Send Email button caption.
- gbTimePoint – Group box for time point selection.

DataGridView Labels

- dgvTitle – DataGridView caption (title).
- dgvColumns – Column headers.
- dgvAddColumn – Added column texts.
- dgvLinkText – Hyperlink column texts.

New Question Group Labels

- buttonNQG – Button caption for new question group.
- buttonNQGtooltip – Tooltip text for new question group button.
- columnName = first field in the group.

Syntax for dgvAddColumn:

addedColumnText:=text/case statement

Example:

IDLang	IDLabel	LType	Ttext
SRC	1	dgvAddColumn	Попуњена пописница:=case when isnull(p1.Podatak,0) in (1,2,3) then '✓' else '' end;
SRL	1	dgvAddColumn	Popunjena popisnica:=case when isnull(p1.Podatak,0) in (1,2,3) then '✓' else '' end;
EN	1	dgvAddColumn	Completed census questionnaire:=case when isnull(p1.Podatak,0) in (1,2,3) then '✓' else '' end;

Syntax for dgvLinkText:

link text Separated By**$**char

Example:

IDLang	IDLabel	LType	Ttext
SRC	1	dgvLinkText	Измени$Пописница
SRL	1	dgvLinkText	Izmeni$Popisnica

EN	1	dgvLinkText	Edit$Questionnaire P1

Syntax for dgvColumns:

ColumnHeaderText1= column1,ColumnHeaderText2=column2,ColumnHeaderText3=column3 etc.

Example:

IDLang	IDLabel	LType	Ttext
EN	1	dgvColumns	Ordinal number of the apartment = rbrstan, Type of apartment = Vrstastana, Number of apartments on the door = BrojStana, Area of apartment = Povrsina, Number of rooms in apartment = BrojSoba, Kitchen in apartment = Kuhinja, Kupatilo
SRC	1	dgvColumns	Редни број стана=rbrstan,Врста стана=Vrstastana,Број стана на вратима=BrojStana,Површина стана=Povrsina,Број соба у стану=BrojSoba,Кухиња у стану=Kuhinja,Купатило у стану=Kupatilo
SRL	1	dgvColumns	Redni broj stana=rbrstan,Vrsta stana=Vrstastana,Broj stana na vratima=BrojStana,Površina stana=Povrsina,Broj soba u stanu=BrojSoba,Kuhinja u stanu=Kuhinja, Kupatilo u stanu=Kupatilo

Error Reporting, Forensics, And Audit Trail- Tables Structure Definitions

Table 1: _ISTLogDelete

Field	Type	Description
appCode	char (8)	IST app code
tableName	nvarchar (25)	table name
user_delete	nvarchar (50)	User's account that has deleted row
date_delete	Datetime	Date and time of delete
primaryKey	nvarchar (MAX)	Primary key of record that is being deleted

Table 2: _ISTLogBatchLC

Field	Type	Description
appCode	char (8)	IST app code
ISTYEAR	char (4)	Year from IST
ISTMONTH	char (2)	Month from IST
tableName	nvarchar (25)	table name
errNumber	varchar (18)	Error number
Rows	Int	Number of rows with errors
Error	nvarchar (4000)	Error
errTitle	nvarchar (255)	Description of the error

Action	nvarchar (255)	The action that goes with the error
user_insert	nvarchar (50)	User's account that did batch logical control
date_insert	Datetime	Date and time of insert

Table 3: _ISTVariablesChange

Field	Type	Description
appCode	nvarchar (8)	IST app code
ISTYEAR	char (4)	Year from IST
ISTMONTH	char (2)	Month from IST
tableName	nvarchar (25)	Table name
primarykey	nvarchar (255)	Primary key of changed record
Variable	nvarchar (50)	Variable
typeChanges	char (50)	IST_Edit, IST_Input
user_insert	nvarchar (50)	User's account that made input or edit
date_insert	Datetime	Date and time of insert

Table 4: _ISTLogExcelXMLJSON

Field	Type	Description
date_insert	Datetime	Date and time of insert
user_insert	nvarchar (50)	User's account that made input or edit
ExcelXMLJSON	nvarchar (5)	Excel / XML / JSON type
Query	nvarchar (MAX)	Query behind generated output file
ISTmodule	nvarchar (100)	IST module that generated output file

Table 5: _ISTLogProcessUsage

Field	Type	Description
appCode	nvarchar (8)	IST app code
ISTYEAR	char (4)	Year from IST
ISTMONTH	char (2)	Month from IST
Process	nvarchar (255)	IST module
Report	nvarchar (255)	IST report
Query	nvarchar (255)	IST Query
user_insert	nvarchar (50)	User's account
date_insert	Datetime	Date and time of insert

Table 6: _ISTSavedAdvancedSearch

Field	Type	Description
date_insert	Datetime	Date and time of insert
user_insert	nvarchar (50)	User's account
Title	nvarchar (255)	Title
Remark	nvarchar (500)	Remark
Query	nvarchar (MAX)	query
appCode	nvarchar (8)	IST app code
condition	nvarchar (MAX)	Saved condition

6.5 Performance Best Practices

Best Practices & Recommendations for IST Developers

Centralized File Management

It is recommended to maintain a single external repository for all queries, programs, headers, macros, and related files. This approach simplifies organization, facilitates regular backups, and ensures that all developers work with consistent versions. For example, in SORS a dedicated file server is used, with a dedicated directory IST\SQLtxt, which is regularly backed up.

Performance Optimization

To optimize IST performance during data entry and validation, consider the following recommendations:

- On form fields, use drop-down lists (preferably AutoCompleteTextBox) instead of relying on functions such as Relacija, WebRelacija, WebOnlyRelacija.
- Define field types and lengths in metadata (_ISTTablesColumns) to avoid redundant validation checks.
- Configure enabling/disabling logic directly in metadata rather than in validation rules.
  Use prefixes (MEM., HEAD., TABLE.) to reference RAM values and avoid unnecessary database queries.
- Write efficient SQL queries for virtual fields and logical control rules. IST executes rules exactly as defined, so rule performance depends on SQL execution time. Test queries independently on the server to identify bottlenecks
- Switch to record- level validation mode (instead of field-level) where appropriate by setting the corresponding parameter (T1 field in _ISTTables metadata). This runs validation on the entire record instead of after every field exit.
- Always design validation rules to prevent errors at the point of data entry, keeping checks lightweight and efficient.

Error Prevention at Data Entry

Preventing errors as early as possible improves both speed and quality. Developers should:

- Always set compulsory keys in metadata to ensure unique identification.
- Configure metadata-driven rules for skipping, visibility, and enabling/disabling fields.

- Provide clear default values where applicable to guide interviewers.
- Where feasible, replace manual validation rules with drop-down lists (combo boxes).

Audit Trail & Logging

The DEPO database on SQL Server is used as a secure, append-only audit log. It is accessible to all users but only allows INSERT actions. Deletions are not possible without administrative permission. Developers should rely on the following logs for full traceability:

- _ISTLogDelete – tracks all deleted records, including survey, table, key, and user.
- _ISTLogLC – records batch logical control executions.
- _ISTChangeVariables – logs changes to variables at save events.
- _ISTLogProcessUsage – records all executed reports and procedures.
- _ISTLogExcelXMLJSON – stores queries behind exported outputs.
- _ISTSavedAdvancedSearch – stores advanced search queries.

These logs support transparency, reproducibility, and GSBPM 5.1 compliance.

Multi-language Support

IST supports multi-language metadata through tables such as _ISTLabels, _ISTMessages, _ISTAWLs, and _ISTQuestionExplanation. Developers should:

- Always provide translations for labels, messages, and explanations in supported languages.
- Keep terminology consistent across applications.
- Ensure that new fields and forms include translations from the start to avoid gaps later.

Metadata Inheritance

The _IST table supports inheritance to reduce duplication and ensure consistency across applications. Developers are encouraged to:

- Use inheritance when multiple applications share identical structures or rules.
- Document clearly when an application inherits from another.
- Avoid unnecessary duplication by reusing tables, rules, and report definitions.
- Test inherited configurations carefully to ensure they function as expected in the new application.

Practical Tips For Boosting The Performance Of Ist Applications

The following guidelines provide best practices for optimizing the performance of IST applications. They cover both MS SQL database design and the effective use of IST metadata.

MS SQL Database Best Practices

- Always define a Primary Key
  Each table should include a primary key.
- Keep row size under 8,000 KB
  Ensure that a single row does not exceed 8,000 KB, so that each row can fit on one memory page.
- Reduce the number of columns
  Fewer columns result in smaller row sizes, allowing more rows per page and reducing I/O overhead when accessing table data.
- Consider table splitting
  If rows are too large, split the table into multiple related tables using a 1–1 relationship.

- Choose appropriate data types
    - Use the narrowest possible data type for each field.
    - If only English text is stored, use varchar or char instead of nvarchar or nchar.
    - For small numeric sets, avoid bigint.
    - Use tinyint for values from 0–255, smallint for -32,768 to 32,767, and int for larger ranges.
    - Use smallmoney instead of money if the range fits (-214,748.3648 to 214,748.3647).
- Avoid deprecated data types
  Replace text/ntext with varchar/nvarchar whenever possible.
- Avoid unnecessary Unicode storage
  Use char/varchar instead of nchar/nvarchar when Unicode is not required.
- General SQL optimization
  Apply all standard SQL Server optimization techniques to queries, indexing, and schema design to improve overall performance.
- Always include INDGR to track valid/invalid records during editing.
- Use LK and LKWeb consistently to enforce locking in both desktop and web applications.
- Implement audit fields (date_INSERT, user_INSERT, date_UPDATE, user_UPDATE) for traceability.
- Add ISTDuration to measure workload and monitor field performance.
- Keep large tables optimized by splitting them into page-sized rows and enforcing primary key constraints for fast access.

DON'T FORGET:

IST applications connect to SQL Server using Windows Authentication. Access is controlled entirely by roles and permissions set on the SQL Server instance - no separate credentials are managed in IST. Developers should make sure required database roles are granted before running or testing IST applications.

IST Metadata Best Practices

- Use simple field validation
- Implement data validation on fields using the stop command instead of relying on _ISTRulesLogicalControl for basic checks.
- Leverage Virtual Fields
    - Use virtual fields (virtualFields) to calculate derived values.
    - Apply the :PK parameter so the virtual field is recalculated only when primary key fields trigger the onValidating event.
    - Use the: CalculateOnExit=nameOfControl(field) parameter to recalculate the virtual field only when the specified control is validated.
- Optimize SQL in virtual fields
    - Remove unnecessary columns from #{select SQL statement}.

    - Optimize all queries for performance.
    - For combo boxes and auto-complete text boxes, keep SELECT statements as narrow as possible.
- Favor EXISTS over NOT IN
  Use EXISTS / NOT EXISTS instead of IN / NOT IN subqueries for better performance.
- Use memory aliasing
  Optimize query performance by using the MEM alias in #{select SQL statement} for virtual fields, which allows data to be retrieved directly from RAM.
- Prefer AutoCompleteTextbox over ComboBox
- Use AutoCompleteTextbox for better performance in large datasets.
- Organize controls with groups - Apply the newQuestionGroup parameter to divide form controls into logical blocks for improved usability.
- Apply IST-specific functions and metadata
    - Use IST functions for data validation in the validatingEvent column of _ISTTablesColumns.
    - Define IST column types and column lengths in _ISTTablesColumns to enforce proper data type validation.
- Always design validation rules to prevent errors at the point of data entry, keeping checks lightweight and efficient.

DON'T FORGET:

- Each IST application is linked to one or more relational databases.
- IST itself does not store or retain statistical data. All individual and aggregated data related to surveys are stored in separate relational databases, which may reside on one or multiple servers within the NSI infrastructure.
- IST optimizes speed by working with a single record at a time in data entry or data editing - this keeps forms responsive even in large databases. For batch validation and automatic corrections, IST switches to batch mode, pulling a set of records from the database and applying rules across them. Developers should keep this two-mode processing model in mind: fast single-record entry/editing vs. controlled bulk operations.
- The quality of IST applications depends on the quality of the relational database design.
- Databases may be hosted on different servers, with connections defined in the _ISTDatabaseConnStrings view.
- Unlike standard SQL server rules, IST places no limitations on table record length. Performance depends primarily on SQL Server.
- IST works record-by-record (via keys), so good database practices (indexing, splitting large tables into page-sized rows, key constraints) improve overall speed.

Performance Optimisation Checklist For Ist Applications

MS SQL Database

☐ Define a primary key for every table.
☐ Keep row size < 8,000 KB (one row per memory page).
☐ Minimize the number of columns per table.
☐ Split large rows into multiple tables with 1–1 relationship.

☐ Choose the narrowest appropriate data type:
- ☐ - tinyint → values 0–255
- ☐ - smallint → values -32,768 to 32,767
- ☐ - int → values -2,147,483,648 to 2,147,483,647
- ☐ - bigint → values -214,748.3648 to 214,748.3647
- ☐ - smallmoney → monetary values -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
- ☐ - decimal → values decimal [(precision [, scale])] for example decimal(10, 2)

☐ Use varchar/nvarchar instead of deprecated text/ntext.
☐ Use char/varchar if Unicode is not required.
☐ Apply general SQL performance best practices (indexes, query optimization, schema design).

IST Metadata

☐ Use field-level validation with stop instead of _ISTRulesLogicalControl for simple rules.
☐ Define virtual fields when derived values are needed.
☐ Apply :PK to recalculate only on primary key validation.
☐ Use: CalculateOnExit=nameOfControl(field) to recalculate only on specific controls.
☐ Remove unnecessary columns from:
- ☐ - #{select SQL statement} for virtual fields
- ☐ - data sources for comboBox and autoCompleteTextBox

☐ Optimize all select statements.
☐ Use EXISTS/NOT EXISTS instead of IN/NOT IN.
☐ Use the MEM alias in #{select SQL statement} to pull data directly from RAM.
☐ Prefer autoCompleteTextBox over ComboBox for large datasets.
☐ Use newQuestionGroup to divide controls into blocks.
☐ Apply IST functions for validation in validatingEvent (_ISTTablesColumns).
☐ Always design validation rules to prevent errors at the point of data entry, keeping checks lightweight and efficient)
☐ Define IST column types and column lengths in _ISTTablesColumns for validation.

Optional Fields Supported In Data Tables (Not Part Of Ist Metadata)

- INDGR (bit) – Marks valid/invalid records; recommended in all tables.
- LK (bit) – If set to 1, record is locked (view-only, no edits or deletes). If all records in time reference are locked, no new inserts are allowed.
- date_INSERT (datetime) – Date record was inserted.
- user_INSERT (nvarchar(50)) – Username of the person who inserted the record.
- sourceOf_INSERT (nvarchar(5)) – Marks data source: I (IST) or W (web application).
- date_UPDATE (datetime) – Date of last record update.
- user_UPDATE (nvarchar(50)) – Username of the person who last updated the record.
- date_LKO (datetime) – Date record was unlocked.
- user_LKO (nvarchar(50)) – Username of the person who last unlocked the record.
- date_LKZ (datetime) – Date record was locked.
- user_LKZ (nvarchar(50)) – Username of the person who last locked the record.

- LKWeb (bit) – If set to 1, record is locked in Web app. Works independently of LK. Web app proposes the next free time reference.
- date_LKOWeb (datetime) – Date record was unlocked in Web app.
- user_LKOWeb (nvarchar(50)) – Username of the person who last unlocked the record in Web app.
- date_LKZWeb (datetime) – Date record was locked in Web app.
- user_LKZWeb (nvarchar(50)) – Username of the person who last locked the record in Web app.
- ISTNote (nvarchar(x)) – Free text field for developer-defined notes.
- QC (tinyint) – Quality control indicator.
- ISTDuration (int) – Time in seconds from record opening to saving.
- Error Indicators (bit) – One per error type defined in IST metadata; identifies records with specific errors.

⚠️ Note: These fields are NOT inserted in IST metadata but may be added in data tables to support editing, quality control, and record locking.

7. Installation and Configuration

7.1 Standard Installation

IST does not require a traditional installation process.
The application is executed directly from the shared network server, and each user workstation connects to the program executable remotely.

Steps:

Create a shortcut to the program launcher located on the central server, for example:

\\serverName\IST\ProcessStarter.exe

or copy the shortcut IST.lnk to the desktop of the user's PC.

The launcher (ProcessStarter.exe) determines which IST version to run.
The file ProcessStarter.exe.config specifies the path to the active executable version (e.g., IST4.NET\Istrazivanja.exe).

To enable version control, up to six concurrent IST versions can be stored in the server directory (IST1, IST2, ..., IST6), allowing maintenance and testing without interfering with the production environment.

Example configuration:

```
<?xml version="1.0" encoding="Windows-1252"?>
<configuration>
 <appSettings>
  <add key="AppPath" value="\\Rzsftp\IST\IST4.NET\Istrazivanja.exe"/>
 </appSettings>
</configuration>
```

File Structure

Each IST version folder contains the main executable, and all required dynamic link libraries (DLLs), macros, and configuration files.
Typical directory contents:

- Istrazivanja.exe – main executable
- Istrazivanja.exe.config – connection and environment configuration
- DLL components (e.g., IstrazivanjaUnos.dll, MyDataGrid.dll, ISTUI.dll, etc.)
- Office interoperability files (for Excel reporting)
- Macro templates: ISTMacro.xls, ISTmPivotChart.xlsm

The configuration file defines connection parameters for all components such as:

- Metadata database (IST),
- Data warehouse (DEPO),
- CATI/CAPI databases,
- External folders for SQL scripts, macros, and documentation.

7.2 Regional Office Installation

Regional offices typically run IST locally to reduce network load.
For these setups, the utility ISTCheckThanStart.exe ensures version consistency:
When a user starts IST, the utility checks whether the local version in %Temp%\IST matches the current version on the server.

If outdated, the new version is automatically copied to the local folder.
This guarantees all users run the same executable, even when operating offline.

7.3 Configuration Details

Each instance of Istrazivanja.exe.config contains connection sections for modules such as:

<ISTconnection> – connection to metadata database
<ISTDeletings> – configuration for DEPO logs (deletion tracking)
<ISTCATI> and <ISTCAPI> – specialized databases for telephone and field data collection
<Istrazivanja> – UI and operational settings such as:
FolderZaProgrameZaUnos (path for entry modules)
HelpFile (user help path)
folderZaGen (location of SQL, XML, and macro files)
mode (DESKTOP, CATI, or CAPI mode)
defaultLanguage and office (language and regional setup)

Parallel Execution and Resource Management

Multiple IST instances can run simultaneously on a single workstation.
Each instance maintains its own RAM allocation and SQL connection, ensuring independent operation.
Connections are automatically opened and closed per record to maximize concurrency and performance.
All metadata and user activity logs are safely stored on the central SQL Server.

8. Security and Access Control

8.1 Authentication & Single Sign-On

Authentication and Database Access

IST uses Windows Authentication to provide secure access to SQL Server databases. All applications developed within the IST framework operate under the user roles, permissions, and authorization rules defined on the SQL Server instance. This approach ensures consistent enforcement of access control and data protection, fully aligned with institutional IT security policies.

8.2 User Roles and Permissions

IST applications connect to SQL Server using Windows Authentication. Access is controlled entirely by roles and permissions set on the SQL Server instance - no separate credentials are managed in IST. Developers should make sure required database roles are granted before running or testing IST applications.

8.3 Logging & Auditing

The DEPO database on SQL Server serves as the central, secure audit repository for IST applications. It is designed as an append-only system, ensuring that every logged action is permanently recorded. While all users can access the database for transparency, only INSERT operations are permitted, preventing alterations or deletions without explicit administrative approval.

Through dedicated log tables, DEPO provides full traceability of user actions and system processes, supporting transparency, reproducibility, and compliance with GSBPM 5.1. Developers can rely on these logs to monitor record deletions, batch validations, variable changes, report executions, and exported datasets.

9. Integrations

9.1 Microsoft Office Integration

IST applications provide seamless integration with Microsoft Office tools, primarily Excel and Word, to support data analysis, reporting, and dissemination workflows. This integration ensures that users can benefit from familiar Office environments while maintaining the robustness and consistency of IST metadata-driven processes.

Key Features

- Excel Export with Headers
    - Query results from IST are exported into Excel files, matched with predefined header templates (XML, XLS, XLSX, XLSM).
    - All formatting, formulas, and styles defined in the header file are preserved in the output file.
    - Support for macros: if an associated Excel macro exists, it is automatically executed after the dataset export.
- Dynamic Parameters in Headers

    - IST supports variable substitution in header templates using survey time points (e.g., =GGG, =MMM).
    - Developers can design flexible templates where headers automatically update based on the selected period.

- Office Macros and Automation
    - Developers can attach macros to automate repetitive tasks (e.g., formatting, pivot creation, chart generation).
    - IST automatically triggers the macro execution after writing data to the Excel output file.

- Word Integration for Reporting
    - Metadata and datasets can be exported into Word templates, enabling automated report generation.
    - Word templates can include bookmarks or placeholders that IST replaces with live data at runtime.

- Standardised Repository
    - All supporting files (SQL scripts, macros, header templates) are stored in the central directory defined by the pathToQuery field in the _IST metadata table.
    - This centralisation ensures easier backups, maintenance, and consistent environments across teams.

Developer Notes

- Always keep header templates in XML Spreadsheet 2003 format where advanced formatting is required.
- Ensure macros are tested independently in Excel before linking them to IST processes.
- Use the ISTASA alias in SQL queries for additional formatting instructions (bold, italics, font size, row height) that propagate directly into Excel outputs.
- Regularly back up the central SQLtxt directory (or equivalent) where macros and headers are stored.

9.2 OpenOffice / LibreOffice Integration

In addition to Microsoft Office, IST also supports integration with OpenOffice and LibreOffice environments, ensuring flexibility for institutions that rely on open-source office suites. This allows statistical applications to be used across a wider range of IT infrastructures, independent of proprietary software.

Key Features

- Spreadsheet Export (Calc)
    - Query results from IST can be exported into ODS (OpenDocument Spreadsheet) format.
    - Like Excel integration, ODS exports can include predefined templates where formatting, formulas, and styles are preserved.
    - IST fills data into these templates at runtime, keeping consistency across reports.

- Dynamic Parameters in Headers

    - Time point parameters such as GGG (year) and MMM (month) can be inserted into ODS headers.
    - Developers can design Calc templates that automatically adapt labels and titles based on the selected survey period.

- Macros and Automation
    - OpenOffice/LibreOffice macros (written in LibreOffice Basic, Python, or JavaScript) can be attached to exported reports.
    - IST triggers macro execution after data is written to the ODS file, enabling automated formatting, charting, or summarisation.

- Writer Integration for Reporting
    - Data and metadata can be exported to ODT (OpenDocument Text) templates for narrative reports.
    - Placeholders or bookmarks in Writer templates are replaced with live IST data at runtime, producing ready-to-use documents.

- Standardised Repository
    - As with Microsoft Office integration, all supporting files (queries, macros, templates) are stored centrally in the folder specified in the _IST.pathToQuery metadata field.
    - This guarantees version control, consistent backups, and uniform outputs across users.

Developer Notes

- Use ODS templates for complex formatting and ensure they are thoroughly tested before deployment.
- Macros in OpenOffice/LibreOffice must be enabled in the local office suite's security settings. Always document macro logic for reproducibility.
- Keep templates lightweight: avoid unnecessary styling or large embedded objects that may slow down exports.
- Regularly back up the central directory with ODS/ODT templates and macros to ensure disaster recovery and environment consistency.

9.3 QGIS (Geospatial Integration)

The IST platform integrates with QGIS, a free and open-source Geographic Information System (GIS), to enhance data collection and fieldwork by providing spatial context for addresses and survey units.

Key Features

- Address Mapping
    - Enumerators can open a QGIS map with a single click.
    - The map displays all addresses assigned to them.
    - The map is dynamically connected to the enumerator's address book - any changes in the address book are immediately reflected on the map, and updates on the map are written back to the address book.

- User-Friendly Adaptation

    - QGIS was adapted and simplified to ensure ease of use by enumerators.
    - The interface was streamlined for quick access to essential functions.

- Automation with PyQGIS
    - Python scripts (PyQGIS) are used to automate tasks within QGIS.
    - Custom plugins were developed to extend functionality and integrate tightly with IST workflows.
    - The QGIS Python API enables data manipulation, spatial analysis, and the creation of tailored user interfaces.

Developer Notes

- Maintain compatibility between IST metadata address book tables and QGIS project layers to ensure synchronization.
- Test PyQGIS scripts for performance, especially when handling large spatial datasets.
- Store and version-control custom QGIS plugins/scripts centrally for maintainability.
- Ensure enumerators' maps are lightweight (filtered by assignment) to improve usability in the field.

9.4 Power BI (Analytics & Dashboards)

IST applications can integrate with Power BI to deliver advanced analytics, interactive dashboards, and visual storytelling. From the IST Reports section, users can open Power BI dashboards directly, combining statistical outputs with modern BI features.

Key Features

- Direct Launch from Reports Section
    - Power BI dashboards can be registered in the IST ReportsProcedures metadata table.
    - Selecting a report in IST can automatically open the linked Power BI dashboard (via URL or embedded frame).

- Dynamic Parameters
    - IST time point parameters (ISTYear, ISTMonth) can be passed into Power BI reports, enabling period-based filtering.
    - Custom parameters defined in IST (e.g., region, sector, demographic group) can also be injected into Power BI queries.

- Data Source Integration
    - IST can export datasets to SQL Server views or staging tables, which Power BI connects to for visualisation.
    - Alternatively, IST-generated Excel or CSV outputs can serve as scheduled refresh sources for Power BI dashboards.

- Dashboard Types
    - Operational dashboards: Monitor fieldwork progress, response rates, or validation errors.
    - Analytical dashboards: Explore statistical indicators with filters, slicers, and drill-throughs.

    - Dissemination dashboards: Provide external stakeholders with interactive public-facing data stories.

- Central Repository & Versioning
    - Power BI dashboard URLs, PBIX files, or shared workspaces should be documented in _ISTReportsProcedures.
    - This ensures that report navigation from IST is consistent and traceable.

Developer Notes

- Use parameterised SQL views in SQL Server to streamline Power BI refresh cycles.
- Keep PBIX files under version control (Git or central repository) to maintain history of dashboard changes.
- Define clear roles and permissions in Power BI to align with IST user groups.
- Test dashboards with large datasets to avoid performance bottlenecks; optimise queries at the SQL Server level when possible.
- For dissemination, consider publishing dashboards via Power BI Service or embedding them in a web portal, linked back to IST.

10. Appendices

10.1 Example Configurations

The configuration file Istrazivanja.exe.config

Example Configuration

Below is an example structure of the configuration file:

```
<configuration>
 <configSections>
  <section name="ISTConnection" type="System.Configuration.NameValueSectionHandler" />
  <section name="Istrazivanja" type="System.Configuration.NameValueSectionHandler" />
  <section name="ISTDeletings" type="System.Configuration.NameValueSectionHandler" />
  <section name="ISTCATI" type="System.Configuration.NameValueSectionHandler" />
  <section name="ISTCAPI" type="System.Configuration.NameValueSectionHandler" />
 </configSections>
 <!-- Connection to IST metadata -->
 < ISTConnection >
  <add key="Data Source" value="nameOfSqlServer" />
  <add key="Initial Catalog" value="IST" />
  <add key="User ID" value="obican" />
  <add key="Password" value="obican" />
 </ ISTConnection >
 <!-- Parameters for managing deleted records in DEPO -->
 <ISTDeletings>
  <add key="Data Source" value="nameOfSqlServer" />
  <add key="Initial Catalog" value="DEPO" />
```

```
  <add key="User ID" value="obican" />
  <add key="Password" value="obican" />
  <add key="SaveDeleted" value="true" />
 </ISTDeletings>
 <!-- CATI and CAPI configurations -->
 <ISTCATI>
  <add key="Data Source" value="nameOfSqlServer" />
  <add key="Initial Catalog" value="CATI" />
 </ISTCATI>
 <ISTCAPI>
  <add key="Data Source" value="nameOfSqlServer" />
  <add key="Initial Catalog" value="DEPO" />
 </ISTCAPI>
 <!-- General IST settings -->
 <Istrazivanja>
  <add key="User ID" value="" />
  <add key="Password" value="" />
  <add key="FolderZaProgrameZaUnos" value="\\serverName\IST\UNOS" />
  <add key="HelpFile" value="\\serverName\ist\Help\ist.chm" />
  <add key="pismo" value="" />
  <add key="folderZaGen" value="\\serverName\IST\SQLtxt\IST" />
  <add key="mode" value="DESKTOP" />
  <add key="office" value="MAKSTAT" />
  <add key="nonavailable" value="61" />
  <add key="defaultLanguage" value="mkd" />
 </Istrazivanja>
</configuration>
```

10.2 Detailed Explanation of Functions with Examples

ASSIGN

Purpose

Set field values programmatically based on constants, expressions, or other fields. Use it to:

- Pre-fill derived fields
- Normalize user input
- Cascade changes after a user edits a key field
- Enforce defaults and house rules

Assign is side-effect free (no DB writes by itself) and runs within the current form/record context.

Where to Define (Metadata Placement)

Use the narrowest scope that fits the task.

Scope	When it runs	Where to put it	Typical use

Field-level	On control Leave/Validated	_ISTTablesColumns.ValidationEvent for that column	Derive values after a user edits a field
Table-level	During validation or pre-commit	_ISTRulesDataValidation.RuleBody for the table	Normalize/calculated fields before save
Form events	On OnSaveNext, OnExit, OnReturn	_ISTFormEvents.EventBody for the form/event	Finalize values around form lifecycle actions

Tip: Keep short, high-frequency logic at field-level; heavier rules at table/form scope.

Syntax

Minimal

#{AssignTo=TargetField; AssignWhat=ExpressionOrLiteral;}

Guarded

#{AssignTo=TargetField; AssignWhat=ExpressionOrLiteral; AssignIf=BooleanCondition;}

Multi-assign (parallel lists; $ pairs items by position)

#{
AssignTo=Target1$Target2$Target3;
AssignWhat=Value1$Value2$Value3;
AssignIf=Condition1$Condition2$Condition3;
}

Notes

- AssignIf is optional. If omitted, the assignment always executes.
- With multi-assign, list lengths must match. Empty entries are allowed only when intentionally skipping a slot.
- Right-hand sides may reference:
  - Form fields: d.FieldName
  - Virtual fields (FP): #FP{fpName}
  - Timepoint tokens: {GGG}, {MMM}, {GGG±n}, {MMM±n}
  - Constants: 'TEXT', numeric literals, '' (empty), null (DB null)

Parameters

- AssignTo (required): One or more destination fields. Must exist, be writable, and mapped in the form.
- AssignWhat (required): Expression or literal that evaluates to the destination's data type.
- AssignIf (optional): Boolean condition controlling execution. Supports the same expression grammar as StopIf.

Common operators: =, <>, >, >=, <, <=, IN (...), LIKE, IS NULL, IS NOT NULL, AND, OR.

Execution Model

1. Trigger
   - Field-level: when user leaves the control (after validation).
   - Table/form-level: on validation and/or save path; for OnSaveNext/OnExit/OnReturn, at that event.
2. Ordering
   - Within a block: top-to-bottom evaluation.

- Within the same scope: rules are executed in their metadata order.
- Recommended order across functions in a single block: Assign → Stop/Message → Skip/Delete/Exec.

3. Re-entrancy
    - Assign changing another control does not re-fire that control's validation automatically. If you need downstream evaluation, place dependent logic in a common later scope (e.g., table-level).

Data Types & Formatting

- Strings: wrap with single quotes 'text'. To embed a quote, double it: 'O''Brien'.
- Empty vs Null: '' = empty string; null = database null. Choose intentionally.
- Numbers: culture-agnostic; use dot or project's configured decimal separator consistently.
- Dates: use project's accepted format or ISO (YYYY-MM-DD) if supported by your engine bindings.
- Type coercion: destination type wins - invalid coercions raise validation errors.

Interactions with Other Functions

- Skip: Prefer to Assign before Skip, so navigated targets see the updated values.
- Relacija/WebRelacija: Relacija may fill the same fields. If you need Relacija's values to win, run Assign first but guard with AssignIf=false when Relacija hits; or use Relacija's alwaysfromcodebook.
- Delete: Assign should not be used to trigger destructive behavior. Use Delete for row removal.
- Exec: Use Exec for persistence or cross-table effects; Assign only sets values in the current record.
- Visible/Enable/ReadOnly/Color: Assign can feed conditions used by these UI functions; keep responsibilities separate.

Examples

1) Simple default

#{AssignTo=Status; AssignWhat='NEW'; AssignIf=isnull(Status,'')='';}

2) Derive year/month from a date

#{
AssignTo=Year$Month;
AssignWhat=year(d.DocDate)$month(d.DocDate);
AssignIf=d.DocDate IS NOT NULL;
}

3) Mirror to multiple fields

#{
AssignTo=OpsCode$OpsTitleLat$OpsTitleCir;
AssignWhat=d.mbops$#FP{fpOpsTitleLat}$#FP{fpOpsTitleCir};
}

4) Conditional normalization

#{AssignTo=IsMinor; AssignWhat=1; AssignIf=cast(isnull(Age,0) as int)<18;}

5) Clear dependent fields when key changes

#{
AssignTo=ProductCode$ProductTitle;
AssignWhat=d.ProductCode$null;
AssignIf= d.ProductCode<>isnull(#FP{fpLastProductCode},'');
}

6) Compose tokens

#{AssignTo=TimePoint; AssignWhat='{GGG}-{MMM}';}

Best Practices

- One purpose per block: keep blocks focused and readable.
- Guard thoughtfully: protect derived IDs/keys with AssignIf to prevent overwrites.
- Avoid heavy computation on field-level rules - move to table/form scope.
- Name harmony: ensure AssignTo fields exist in _ISTTablesColumns for the table and are on the form.
- Document intent: a short comment line above complex expressions saves time for reviewers.

Common Errors & Diagnostics

Symptom	Likely cause	Fix
"Unknown field"	AssignTo contains a non-mapped or misspelled field	Verify _ISTTablesColumns and form binding
"Type mismatch"	AssignWhat expression result doesn't match destination type	Cast/convert or adjust literal
"List length mismatch"	AssignTo/AssignWhat have different $ counts	Align lists 1:1
Value not applied	AssignIf false or rule runs earlier/later than expected	Recheck guard and scope/order
Overwritten by Relacija	Relacija populates same field after Assign	Change order/guards or use alwaysfromcodebook

Debug tips

- Temporarily log resolved expressions (values of AssignWhat after evaluation).
- Use a "diagnostic label" or form console to show FP tokens ({GGG}, {MMM}, etc.) at runtime.

Performance Guidance

- Consolidate related assignments into a single block to avoid repeated parsing/evaluation.
- Cache/compute expensive FP values once at form load or table-level, not per field leave.
- Avoid string-heavy or SQL-like parsing inside AssignWhat.

Security & Integrity

- Don't assign into audit/system columns unless explicitly intended.
- Respect read-only policy: some projects allow Assign to write UI read-only fields (by design); document your project's stance and be consistent.

- Avoid assigning primary keys after the record is persisted unless the workflow demands it (prefer a controlled “rekey” flow).

Testing Checklist

- Happy path: values apply as expected
- Guard path: AssignIf prevents changes when false
- Data typing: text, numeric, date, '', null
- Order with Skip/Relacija: no unintended overrides
- Field-level vs table-level timing verified
- OnSaveNext/OnExit behaviors correct and persisted
- No infinite interactions (Assign doesn’t re-trigger loops)

Quick Reference (Cheat Sheet)

```
# Single
#{AssignTo=F1; AssignWhat='X';}

# Guarded
#{AssignTo=F1; AssignWhat=d.F2; AssignIf=isnull(d.F2,'')<>'';}

# Multi
#{AssignTo=F1$F2$F3;
AssignWhat='A'$d.B$null;
AssignIf=1=1$d.B>0$d.B IS NULL;}
```

Cross-References

- Skip - navigate after values are set
- Relacija / WebRelacija / WebOnlyRelacija - lookup & assignment from codebooks
- Message - communicate results or warnings to the user
- Exec - perform stored procedures when assignment alone is insufficient

Minimal Metadata Examples

Field-level (in _ISTTablesColumns.ValidationEvent)

```
#{AssignTo=Title; AssignWhat=#FP{fpTitleFromOps}; AssignIf=isnull(d.mbops,'')<>'';}
```

Table-level (in _ISTRulesDataValidation.RuleBody)

```
#{
AssignTo=IsValid$TimePoint;
AssignWhat=case when d.Amount>0 then 1 else 0 end$'{GGG}-{MMM}';
}
```

Form event (in _ISTFormEvents.EventBody for OnSaveNext)

```
#{
AssignTo=SavedAt$SavedBy;
AssignWhat=now()$#FP{fpUser};
}
```

STOP

Purpose

STOPIF evaluates one or more boolean conditions during form interaction. When any condition evaluates to TRUE, the interpreter blocks data entry, keeps focus on the current control, and (optionally) displays a message (STOPMSG) and/or performs companion UI/navigation actions (SKIPTO/SKIPIF, visibility/enabling toggles, clearing/disabling skipped controls). Multiple alternative condition–message pairs can be expressed in a single rule using the $ separator (multi-branch StopIf).

Where to Define (Metadata Placement)

Define STOP-related rules in the metadata tables that host validation and event logic. Use these placements:

- _ISTRulesDataValidation.validationEvent - row-level field validation (most common).
- _ISTTablesColumns.columnAttributes - control-level validators tied to a specific field/control.
- _ISTForms.formEvents - form-level STOPs (e.g., OnSaveNextStopIf, OnExitStopIf).
- _ISTMessages - central repository for reusable messages referenced via StopMsg IDs (ISTMSG*).

Choose the narrowest appropriate scope: column → row/form → cross-form, to keep rules maintainable.

Syntax

Canonical field-level form:

#{ StopIf=CONDITION; StopMsg=MESSAGE_TEXT or ISTMSG_ID; }

Optional focus override (after stop):

#{ StopIf=CONDITION; StopMsg=MESSAGE; StopFocusOn=CONTROL_NAME; }

Multi-branch variant (parallel lists with $ pairing):

#{
StopIf=COND_1$COND_2$COND_3;
StopMsg=MSG_1MSG_2MSG_3;
}

IST message-table reference:

#{StopIf=CONDITION; StopMsg=ISTMSG123; }

Specialized forms on buttons/events:

#{
OnSaveNextStopIf=CONDITION; OnSaveNextStopMsg=MESSAGE;
[OnSaveNextStopMsgFocusOn=CONTROL];
}
#{OnExitStopIf=CONDITION; OnExitStopMsgIs=MESSAGE; [OnExitStopMsgFocusOn=CONTROL];}

Panel-based custom message (modal):

#{ StopMsgOnPnl=MESSAGE or ISTMSG_ID; StopIf=CONDITION; }

Parameters

- StopIf - Boolean expression. When true, execution halts and a message is shown.
- StopMsg / StopMsgIs - Text to display, or an ISTMessages identifier (e.g., ISTMSG50).
- StopMsgOnPnl - Shows a custom panel with the message instead of a standard dialog.

- StopFocusOn / On*StopMsgFocusOn - (Optional) control to focus after the message is dismissed.
- OnSaveNextStopIf / OnExitStopIf - Event-scoped conditions for SaveNext/Exit actions.

Execution Model

- Evaluated at the point of the event (e.g., control leave/validate, SaveNext, Exit).
- On true, the engine cancels the originating action (navigation/save/close).
- Message is rendered; focus may be redirected if configured (StopFocusOn).
- No data is committed in the canceled action path.

Data Types & Formatting

- Use CAST/CONVERT wrappers for numeric comparisons (e.g., cast(isnull(A,0) as int) > 0).
- Use ISNULL to guard against NULLs in expressions.
- String comparisons require quotes and proper trimming if needed (e.g., RTRIM(LTRIM(str))).
- Multi-branch ($-paired) lists must have the same item count across StopIf and StopMsg.

Interactions with Other Functions

- Skip - STOP cancels navigation even if Skip rules are present.
- Assign - STOP is evaluated before Assign side effects; assignments are not applied when stopped.
- Delete - STOP prevents DeleteFrom/DeleteWhere/If execution in the same event block.
- Exec - Stored-proc calls in the same event are not executed when STOP triggers.
- Message - Prefer StopMsg for hard validation; MsgIs/SimpleMsgIs for informational notices.

Examples

Single-stop with inline text:

```
#{
StopIf=cast(isnull(AGE,0) as int) < 18;
StopMsg='Mora biti 18+.';
}
```

Using ISTMessages:

```
#{
StopIf=isnull(status,0)=0;
StopMsg=ISTMSG50;
}
```

Multi-branch:

```
#{
StopIf=isnull(A,0)=0$isnull(B,0)=0;
StopMsg='A je obavezan.'$'B je obavezan.';
}
```

On SaveNext:

```
#{
OnSaveNextStopIf=isnull(#FP{fpValid},0)=0;
OnSaveNextStopMsg='Provera nije prošla.';
OnSaveNextStopMsgFocusOn=CONTROL_X;
}
```

On Exit with custom panel:

```
#{
OnExitStopIf=hasPendingErrors=1;
OnExitStopMsgOnPnl='Ispravite greške pre izlaska.';
}
```

Best Practices

- Scope STOP as locally as possible (per-control before form-level).
- Use ISTMessages (ISTMSG*) for reusable/localized text.
- Combine STOP with precise focus guidance (StopFocusOn) to speed correction.
- Group related checks into a single multi-branch block for readability.
- Document each STOP with a short rationale in comments.

Common Errors & Diagnostics

- Mismatched $-branch counts between StopIf and StopMsg.
- Uncast types in numeric comparisons leading to string comparison bugs.
- Referencing non-existent controls in StopFocusOn.
- Overlapping STOPs on the same event causing duplicate dialogs.
- Silent bypass because StopIf references unset fields - add ISNULL guards.

Performance Guidance

- Prefer simple Boolean expressions; avoid heavy SQL calls in StopIf.
- Cache expensive lookups outside StopIf; use Relacija for codebook checks.
- Consolidate multiple STOPs into multi-branch rules to reduce evaluation overhead.

Security & Integrity

- STOP prevents invalid writes; do not ‘downgrade’ to MsgIs for critical rules.
- Do not leak sensitive data in messages; prefer generic wording where needed.
- Keep ISTMessages locked down to authorized editors for consistency.

Testing Checklist

- StopIf triggers on invalid inputs
- Correct StopMsg/ISTMSG text appears
- Cancelled action (no save/close/skip) confirmed
- Focus lands on the right control (if configured)
- Multi-branch pairs aligned and show proper message per case
- Does not conflict with Skip/Assign/Delete/Exec in same event
- Localized/ISTMessages verified
- Edge cases with NULL/empty handled

Quick Reference (Cheat Sheet)

Minimal:
#{StopIf=COND; StopMsg='TEXT';}

Multi-branch:
#{StopIf=C1$C2; StopMsg='M1'$M2';}

Focus:
#{StopIf=COND; StopMsg=ISTMSG50; StopFocusOn=CONTROL;}

OnSaveNext:
#{
OnSaveNextStopIf=COND; OnSaveNextStopMsg=MSG;
OnSaveNextStopMsgFocusOn=CONTROL;
}

OnExit:
#{OnExitStopIf=COND; OnExitStopMsgIs=MSG;}

Cross-References

- Assign - to set defaults after successful validation.
- Skip - to branch flows when inputs are valid.
- Delete - to clean RAM/child rows post-validation (only when not stopped).
- Message - for non-blocking user guidance.
- Relacija - for codebook/data-master validations prior to STOP.

Minimal Metadata Examples
Field-level STOP in _ISTTablesColumns.columnAttributes:
#{
Min=1; Max=2;
StopIf=isnull(Answer,0)=0;
StopMsg=ISTMSG12;
}
Row-level STOP in _ISTRulesDataValidation.validationEvent:
#{
StopIf=isnull(status,0)=0;
StopMsg='Status is required.';
}
Form-level STOP on SaveNext in _ISTForms.formEvents:
#{
OnSaveNextStopIf=cast(isnull(#FP{fpValid},0) as int)=0;
OnSaveNextStopMsg=ISTMSG50;
}

SKIP

Purpose

Where to Define (Metadata Placement)
Use the Skip function family inside the metadata-driven validation/behavior blocks for controls and panels.
Primary locations:
• _ISTTablesColumns.validationEvent - field-level (TextBox, ComboBox, AutoCompleteTextBox, etc.).
• _ISTTablesPanels.panelAttributes (or equivalent) - panel- and form-level transitions.
• _ISTTablesColumns.columnAttributes - for default skip/visibility rules tied to a control definition.
These are written as inline rule blocks beginning with '#{' and ending with '}'.

Syntax
Single target:
#{SkipTo=TARGET_1; SkipIf=COND_1;}
Multi-branch (parallel lists; '$' pairs up items in order):
#{SkipTo=TARGET_1$TARGET_2$TARGET_3;SkipIf=COND_1$COND_2$COND_3;}
With "skipped" behavior flags:
#{
SkipTo=TARGET_1$TARGET_2;
SkipIf=COND_1$COND_2;
SkippedEnabledFalse;
SkippedSetEmpty;
}
Keep some fields on skip (exceptions):
#{
SkipTo=TARGET_1$TARGET_2;
SkipIf=COND_1$COND_2;
SkippedEnabledFalse;
SkippedSetEmptyExcept=FIELD_A,FIELD_B;
}
Conditional exceptions:
#{
SkipTo=TARGET_1$TARGET_2;
SkipIf=COND_1$COND_2;
SkippedEnabledFalse;
SkippedSetEmptyExcept=FIELD_A,FIELD_B;
SkippedSetEmptyExceptIf=COND_EXC;
}

Parameters
SkipTo
Comma- or '$'-separated list of destination control names (or panels). Use '$' to map multiple conditions to multiple distinct targets.

SkipIf
Boolean expressions evaluated against RAM values (d.FieldName), virtual fields (#FP{...}), or constants. Supports SQL-like functions in your environment (CAST, ISNULL, IN, LIKE).
SkippedEnabledFalse
If set, controls skipped past are disabled.
SkippedSetEmpty
If set, controls skipped past are cleared (set to empty/null).
SkippedSetEmptyExcept
Comma-separated list of fields that must not be cleared even when SkippedSetEmpty is active.
SkippedSetEmptyExceptIf
Condition under which the exception list applies.
SkippedEnabledFalseIf
Optional condition controlling when the disablement applies.
SkipAction
Defines what happens to the skipped fields when a skip condition is met. It controls data cleanup and UI state for fields that are bypassed.

Execution Model
• Evaluated immediately upon leaving the current control (or when the rule block is triggered).
• If a matching SkipIf condition is true, the UI focuses the corresponding SkipTo target. When multiple branches are provided, they are evaluated in order with '$' mapping.
• When 'skipped' directives are present, the engine disables and/or clears the intervening controls according to the flags and exceptions.
• Skip rules execute after Stop validations (if Stop triggered, Skip won't execute).
• Skip can coexist with Assign, Visible/Enable, ReadOnly, and others; order of evaluation follows the platform’s standard control event pipeline.

Data Types & Formatting
• Use 'cast(...) as int' for numeric comparisons when fields are stored as text.
• Use 'isnull(field,default)' to avoid null comparison errors.
• String comparisons often rely on literal '1','2', etc.; align to your stored types.
• Multi-branch comparisons allow complex in/not in lists, LIKE, and composite conditions.

Interactions with Other Functions
• StopIf/StopMsg: Stop takes precedence; Skip will not fire if a Stop condition triggers.
• Assign: Use Assign to prefill or normalize values before Skip conditions are evaluated.
• Visible/Enabled/ReadOnly: Skip may be combined with SkippedEnabledFalse and SkippedSetEmpty for consistent state management.
• Delete: Skips often co-occur with Delete to clean dependent rows when branches change.
• Message: You can throw messages just before/after Skip via MsgIs/OptionalMsgIs if needed.

Examples
A. Minimal Skip (single branch)
#{
SkipTo=Q41;

SkipIf=4 not in Pomocno_G4;
}

B. Multi-branch based on helper list

#{
SkipTo=Q41$Q37_7;
SkipIf=6 not in Pomocno_G4$6 in Pomocno_G4;
SkippedEnabledFalse;
SkippedSetEmpty;
}

C. Age-driven branching with Assign

#{
yes=1,2;skipTo=CITIZEN;skipIf=2=2;
skippedEnabledFalse;
assignTo=AGE$AGE;
assignWhat=#fp{FPGAGE}$#FP{FPGBIRTHPASS};
assignIf=BIRTHPASS=1$BIRTHPASS=2;
}

D. Conditional branching across many targets (vehicle example)

#{
skipto=CenaKv1$VrsRb1$VrsTmp1$VrsKont1;
skipif=isnull(VrstVoz,0)=1$isnull(VrstVoz,0)=2$isnull(VrstVoz,0)=3$isnull(VrstVoz,0) not in(1,2,3);
skippedenabledfalse;
skippedsetempty
}

E. Skipping with exceptions (keep some fields)

#{
SkipTo=BrojDom$UBrLicaStan$ButtonNext;
SkipIf=cast(isnull(KoZiviuStanu,0) as int)=1$cast(isnull(KoZiviuStanu,0) as
int)=2$cast(isnull(KoZiviuStanu,0) as int) in (3,4,5,6,7);
SkippedEnabledFalse;
SkippedSetEmptyExcept=BrojDom$BrojDom,UBrLicaStan;
SkippedSetEmptyExceptIf=cast(isnull(KoZiviuStanu,0) as int)=2$cast(isnull(KoZiviuStanu,0) as int) in
(3,4,5,6,7);
}

F. Complex survey grid routing

#{
No=;Yes=1,2;
SkipTo=C3_1_B$C3_1_A1;
SkipIf=cast(isnull(C3_1_A,0) as int)=2$cast(isnull(C3_1_A,0) as int)<>2;
SkippedEnabledFalse;
SkippedSetEmpty;
}

G. Multi-branch plus Stop (validation and routing)

#{
No=;Yes=MultiResponseNumeric;Min=1;Max=9;
StopIf=isnull(C3_2_A,0)=2 and isnull(C3_2_B,0)=2 and isnull(C3_2_C,0)=2 and isnull(C3_2_D,0)=2;
StopMsg='Mora biti označena barem jedna vrsta DRVNIH GORIVA';
SkipTo=C3_2_D1$C3_3_A$C3_4_A;
SkipIf=2=2$2=2$2=2;
}

H. Year-based enabling/locking of address block with Skip

#{
skipto=UGradUt1$UDrzUT1;
skipif=god<2024$god>=2024;
enabledfalse=Mesto,Ulica,NazivPrUsl,UVrsRb1,UVrsTmp1,UVrsKont1,UNapom1,UVrsRb2,UVrsTmp2,UVrsKont2,UNapom2,UVrsRb3,UVrsTmp3,UVrsKont3,Unapom3,UVrsRb4,UVrsTmp4,UVrsKont4,Unapom4,UVrsRb5,UVrsTmp5,UVrsKont5,UNapom5;
}

I. Cascading vehicle price sections (patterned groups)

#{skipto=UCenaKv1;skipif=2=2;skippedenabledfalse;skippedsetempty}
#{
skipto=UCenaKv2$UVrsRb2$UVrsTmp2$UVrsKont2;
skipif=isnull(VrstVoz,0)=1$isnull(VrstVoz,0)=2$isnull(VrstVoz,0)=3$isnull(VrstVoz,0)=4;
skippedenabledfalse;skippedsetempty
}

J. Household roster branching with cleanup (paired with Delete)

#{
SkipTo=BrojDom$UBrLicaStan$ButtonNext;
SkipIf=cast(isnull(KoZiviuStanu,0) as int)=1$cast(isnull(KoZiviuStanu,0) as int)=2$cast(isnull(KoZiviuStanu,0) as int) in (3,4,5,6,7);
skippedenabledfalse;skippedsetemptyExcept=BrojDom$BrojDom,UBrLicaStan;
AssignTo=BrojDom$BrojDom,UBrLicaStan;AssignWhat=0$0;
AssignIf=cast(isnull(KoZiviuStanu,0) as int)=2$cast(isnull(KoZiviuStanu,0) as int) in (3,4,5,6,7);
deleteFrom=P2_Dom,SpisakLica,P1;
deleteIF=cast(isnull(KoZiviuStanu,0) as int)=2 and cast(isnull(#FP{fpCountP2_Dom},0) as int)>0;
deleteWhere=RbrDom>0;
deleteQuestion=true
}

K. Grid-level skip with Enable/Visible coordination

#{
yes=1,2,3;no=;
skipto=BIRTHPLACE;
skipif=isnull(ip6,0)=1;
skippedSetEmpty;
SkippedEnabledFalse;

VisibleFalse=ButtonEHIS_Dnevnik_Anketiranja_Lica;
VisibleFalseIf=isnull(P3,0)<>4 and isnull(P3,0)<>5;
ReadOnly=GRID_LICA;
ReadOnlyAko=isnull(P3,0)>=1 and isnull(P3,0)<=3;
}

L. Textbook conditional parallel branches

#{
No=;Yes=1,2;
SkipTo=C1_1$C2_1_0;
SkipIf=cast(isnull(C1_0,0) as int)=2$cast(isnull(C1_0,0) as int)<>2;
SkippedEnabledFalse;
SkippedSetEmpty;
}

M. Skipping to Save button / end-of-panel

#{notabstop;skipto=buttonsave;skipif=2=2;skippedenabledfalse;}

N. Combined routing with hard validation on prerequisites

#{
skipto=Q4_0;
skipif=isnull(Q3,'')<>'' and isnull(Q3_1,'')<>'' and isnull(Q3_2,'')<>'';
skippedenabledfalse;skippedsetempty;
stopmsg='Wrong!';
stopif=isnull(Q3_1,0)+isnull(Q3_2,0)<50;
}

O. Language/region-specific routing

#{
skipto=PoljePol$Opstina;
skipif=Drzava<>'Република Србија'$Drzava='Република Србија';
skippedenabledfalse;skippedsetempty;
}

P. SkipAction

One-branch example (multiple actions in the same branch):

#{
 SkipTo=NextSection;
 SkipIf=HasChildren=0;
 SkipAction=SetEmpty,EnabledFalse,VisibleFalse,DontSave;
}

Two-branch example (actions aligned to branches):

#{
 SkipTo=A_1_13_4$buttonnext;
 SkipIf=A_1_4_4>2$A_1_4_4 in (1,2);
 SkipAction=SetEmpty,EnabledFalse$DontSave,VisibleFalse;
}

Interpretation: first branch clears and disables; second branch does not save and hides fields.

Per-branch single actions (shorthand):

```
#{
 SkipTo=Q1$Q2$Q3;
 SkipIf=Cond1$Cond2$Cond3;
 SkipAction=SetEmpty$EnabledFalse$VisibleFalse;
}
```

Use $ to separate actions per branch, matching the order of SkipTo/SkipIf branches.
Within a branch, separate multiple actions with commas (,).

Best Practices

- Keep SkipIf conditions mutually exclusive when using multi-branch '$' mapping.
- Always pair SkippedSetEmpty with SkippedEnabledFalse to avoid invisible but active data.
- When clearing, protect retained keys via SkippedSetEmptyExcept.
- For complex grids, prefer smaller, composable rule blocks over one huge block for maintainability.
- Use Delete in the same block if branch changes invalidate dependent child rows.

Common Errors & Diagnostics

- Mismatched '$' counts between SkipTo and SkipIf - ensure 1:1 mapping.
- Referencing non-existent targets - verify control/panel names.
- Type mismatches in SkipIf - normalize with CAST/ISNULL.
- Hidden-but-enabled fields after skip - ensure SkippedEnabledFalse is declared.
- Data lingering after branch change - add SkippedSetEmpty/SkippedSetEmptyExcept.

Performance Guidance

- Prefer simple boolean conditions; pre-normalize with Assign in prior steps.
- Avoid duplicate heavy expressions; factor into #FP virtuals when reusable.
- Break large multi-branch blocks into logical groups to reduce evaluation overhead.

Security & Integrity

- Skip only affects UI flow and RAM values. Persisted data integrity should be protected with server-side checks (Stop/validation) and database constraints.
- When combined with Delete, ensure deleteWhere scopes rows safely (key-bound conditions).

Testing Checklist

[ ] All SkipTo targets exist (controls/panels).
[ ] SkipIf conditions are mutually exclusive (when using '$').
[ ] SkippedEnabledFalse/SkippedSetEmpty behave as intended.
[ ] Exceptions in SkippedSetEmptyExcept are honored.
[ ] Integration with Stop/Assign/Visible/Enable works across paths.
[ ] Dependent child tables cleaned (Delete) when path changes.
[ ] Keyboard navigation still behaves (NoTabStop where needed).
[ ] Edge cases: nulls, empty strings, '0' vs 0, type casts.
[ ] Localization of StopMsg/labels where applicable.

Quick Reference (Cheat Sheet)

Single

#{SkipTo=TGT; SkipIf=COND;}

Parallel

#{SkipTo=T1$T2$T3;SkipIf=C1$C2$C3;}

With skipped behavior

#{SkipTo=T1$T2;SkipIf=C1$C2;SkippedEnabledFalse;SkippedSetEmpty;}

With exceptions

#{SkipTo=T1$T2;
SkipIf=C1$C2;
SkippedEnabledFalse;
SkippedSetEmptyExcept=KEEP1,KEEP2;
SkippedSetEmptyExceptIf=C_EXC;
}

Cross-References

- Assign - normalize/seed values used by SkipIf.
- Stop - guard rails; prevents Skip when conditions fail.
- Visible/Enable/ReadOnly - state control often paired with Skip.
- Delete - cleanup dependent rows on branch change.
- Message (MsgIs/OptionalMsgIs) - prompt users around branch transitions.

Appendix - Extended Example Collection

#{SkipTo=Q41;SkipIf=4 not in Pomocno_G4;SkippedEnabledFalse;SkippedSetEmpty}
#{SkipTo=Q63;SkipIf=cast(isnull(Q61_2,0) as int)<>2;SkippedEnabledFalse;SkippedSetEmpty}
#{SkipTo=Q45_1;SkipIf=1 not in Pomocno_G11;SkippedEnabledFalse;SkippedSetEmpty}
#{skipto=UGradUt1$UDrzUT1;skipif=god<2024$god>=2024;enabledfalse=Mesto,Ulica,NazivPrUsl,...}
#{skipto=UCenaKv1$UVrsRb1$UVrsTmp1$UVrsKont1;skipif=isnull(VrstVoz,0)=1$isnull(VrstVoz,0)=2$isnull(VrstVoz,0)=3$isnull(VrstVoz,0)=4;skippedenabledfalse;skippedsetempty}
#{skipto=UCenaKv2$UVrsRb2$UVrsTmp2$UVrsKont2;skipif=isnull(VrstVoz,0)=1$isnull(VrstVoz,0)=2$isnull(VrstVoz,0)=3$isnull(VrstVoz,0)=4;skippedenabledfalse;skippedsetempty}
#{skipto=ICenaKv1$IVrsRb1$IVrsTmp1$IVrsKont1;skipif=isnull(VrstVoz,0)=1$isnull(VrstVoz,0)=2$isnull(VrstVoz,0)=3$isnull(VrstVoz,0)=4;skippedenabledfalse;skippedsetempty}
#{skipto=GradUt1;skipif=2=2;}
#{skipto=CenaKv1$VrsRb1$VrsTmp1$VrsKont1;skipif=isnull(VrstVoz,0)=1$isnull(VrstVoz,0)=2$isnull(VrstVoz,0)=3$isnull(VrstVoz,0) not in(1,2,3);skippedenabledfalse;skippedsetempty}
#{skipto=mesec1;skipif=2=2;}
#{SkipTo=ip7_Drugo$BIRTHPLACE;SkipIf=isnull(ip7,0)=6$isnull(ip7,0)<>6;SkippedEnabledFalse;SkippedSetEmpty;}
#{No=;SkipTo=B8_6O$C1_0;SkipIf=isnull(B8_6,0)<>2$isnull(B8_6,0)=2;SkippedEnabledFalse;SkippedSetEmpty}
#{No=;Yes=1;SkipTo=C1_1$C2_1_0;SkipIf=cast(isnull(C1_0,0) as int)=2$cast(isnull(C1_0,0) as int)<>2;SkippedEnabledFalse;SkippedSetEmpty}

#{No=;Yes=1,2;SkipTo=C1_2$C2_1_1;SkipIf=cast(isnull(C1_1,0) as int)=2$cast(isnull(C1_1,0) as int)<>2;SkippedEnabledFalse;SkippedSetEmpty}
#{No=;Yes=1,2;SkipTo=C1_3$C2_1_2;SkipIf=cast(isnull(C1_2,0) as int)=2$cast(isnull(C1_2,0) as int)<>2;SkippedEnabledFalse;SkippedSetEmpty}
#{No=;Yes=1,2;SkipTo=C1_4$C2_1_3;SkipIf=cast(isnull(C1_3,0) as int)=2$cast(isnull(C1_3,0) as int)<>2;SkippedEnabledFalse;SkippedSetEmpty}
#{notabstop;skipto=buttonsave;skipif=2=2;skippedenabledfalse;}

DELETE

Purpose

The Delete function is used to remove one or more records from RAM-resident data tables at runtime, typically in response to validation outcomes or state changes on a form. It is executed conditionally, based on DeleteIf rules, and scoped by DeleteWhere predicates. Delete is frequently used to realign child/trace/auxiliary tables after edits to primary keys or controlling variables, or when reverting/shortening a list-length field (e.g., reducing household size).

Where to Define (Metadata Placement)

Define Delete in the validation/behavior metadata attached to the control or event that governs the removal (e.g., in _ISTTablesColumns.ValidationEvent for a field, or in a panel/form-level rule block such as SaveNext/Exit handlers where supported). The directive is part of the same rule stanza as other behaviors (Stop, Skip, Assign, etc.) and is executed when the hosting rule evaluates.

Syntax

#{
deleteFrom=Table1,Table2,TableN;
deleteWhere=<SQL-like where predicate>;
deleteIf=<boolean condition>;
}

Parameters

deleteFrom

Comma-separated list of RAM data tables to delete from. Case-insensitive; no spaces required (spaces tolerated).

deleteWhere

A SQL-like predicate that scopes which rows to delete in each listed table. Use field names from the target table and constants/virtual fields (#FP{...}).

deleteIf

A boolean expression. If it evaluates true, the delete operation is executed. If false (or not present), no rows are deleted.

Optional flags

deleteQuestion=true - If present and true, may prompt or mark a question-driven delete. Behavior can be implementation-specific; retain when your examples include it.

Execution Model

1) Evaluate DeleteIf (if present). If false, Delete exits with no action.
2) Evaluate DeleteWhere against each table in deleteFrom, operating on in-memory copies (RAM).
3) Remove matching rows instantly; UI/derived fields refresh on next redraw/validation cycle.
4) Ordering: Within a single rule block, Delete executes after any Stop conditions short-circuiting the block and in the order declared relative to other directives (Assign/Skip/etc.).

Data Types & Formatting

• Use cast(...) when comparing text-to-numeric (e.g., rbr > cast(#FP{fprbr} as integer)).
• Strings must be quoted; numerics unquoted.
• #FP{...} virtual fields resolve to current on-form values at runtime.
• Expressions support AND/OR/IN/NOT/ISNULL and parentheses for grouping.

Interactions with Other Functions

• Stop/StopMsg: If a StopIf triggers, Delete in the same block will not run.
• Skip: Often used together - Delete clears stale child rows, Skip navigates away from now-irrelevant sections.
• Assign: Consider assigning counters/flags before/after Delete to keep counts coherent.
• Visible/Enable/ReadOnly: Use to lock parts of UI before/after deletes to prevent re-entry inconsistencies.
• OnSaveNext / OnExit: Parallel event-level delete variants exist (OnSaveNextDeleteFrom/If, etc.).

Examples

1) Simple trace cleanup on range change

#{DA=Numeric;Ne=,;Min=1;Max=30;STOPIF=isnull(T010200,'')='';STOPMSG='Недозвољена вредност!';skocina=panelTrace1;skociako=2=2;skippedenabledfalse;
deleteFrom=Trace1,Trace2,Lica;
deleteWhere=rbr>#FP{fprbr};
deleteIF=#FP{fprbrTRACE1}>#FP{fprbr};}

2) Variant with explicit cast on threshold

#{DA=Numeric;Ne=,;Min=1;Max=30;SKOCINA=panelTrace1;SKOCIAKO=2=2;
deleteFrom=Trace1,Trace2,Lica;
deleteWhere=rbr>cast(#FP{fprbr} as integer);
deleteIF=#FP{fprbrTRACE1}>#FP{fprbr};}

3) Conditional delete after outcome selection with form-level read-only/visibility logic

#{DA=1,2,3,4,5,6;
skipto=P90$P4$P9;
skipif=isnull(P3,0)=2$isnull(P3,0)=5 or isnull(P3,0)=6$isnull(P3,0)=1 OR isnull(P3,0)=3 or

isnull(P3,0)=4;
ReadOnly=GRID_LICA;
ReadOnlyAko=isnull(P3,0)>=1 and isnull(P3,0)<=3;
VisibleFalse=ButtonEHIS_Дневник_Анкетирања_Лица;VisibleFalseIf=isnull(P3,0)<>4 and isnull(P3,0)<>5 ;NE=;skippedsetempty;skippedenabledfalse;
AssignTo=P90$P31;AssignWhat=''$#FP{fpP31};AssignIf=isnull(P3,0)=5 or isnull(P3,0)=6$isnull(p3,0)<>0;
stopif=isnull(#FP{fpbrlica},'0')='0' AND (isnull(p3,0)=4 or isnull(p3,0)=5);
stopmsg=Не можете одабрати исход 4 или 5 ако је празан списак лица за ово домаћинство!;
deleteFrom=EHIS_Дневник_Анкетирања_Лица;
deleteWhere=ge1=d.ge1 and ge2=d.ge2 and ge5=d.ge5 and ge6=d.ge6 and rb=d.rb;
deleteIF=cast(isnull(#FP{fpG051},'0') as int)>0 and isnull(P3,0) not in (4,5);
}

4) Cascaded child cleanup when roster size shrinks (with UI reaction)

#{Da=numeric;Ne=;skipto=tableEHISSpisakLica;skipif=2=2;
deleteFrom=ehisspisaklica,ehislica15,ehislica7_14,EHISMerenje;
deleteWhere=rbrclan>cast(BrClan as integer);
deleteIF=cast(isnull(#FP{fpMaxLicaSpisak},'0') as int)>isnull(brclan,0);
VisibleTrue=ButtonEHISSpisakLica;
VisibleTrueIf=isnull(brclan,0)<>0;}

5) Multi-branch conditional deletes with user-facing messaging

#{DA=1,2;NE= ;
AssignWhat=fpNavrGod$'';AssignTo=NavrGod$SamopopunjavanjeU;AssignIf=1=1$cast(isnull(#FP{fpNavrGod},'0') as int)<5;
porukaje=
Попунили сте упитник за лица старости 15 и више година, а затим сте променили податке тако да лице има мање од 15 година. Упитник за лице ће бити обрисан.
$Попунили сте упитник за лице старости 5-14 година, а затим сте променили податке тако да лице нема 5-14 година. Упитник за лице ће бити обрисан.
$Попунили сте образац за мерење, а затим сте променили податке тако да лице има мање од 5 година. Образац за мерење ће бити обрисан.;
porukaako=
cast(isnull(#FP{fpNavrGod},'0') as int)<15 and cast(isnull(#FP{fpPostoji15},'0') as int)=1
$(cast(isnull(#FP{fpNavrGod},'0') as int)<5 or cast(isnull(#FP{fpNavrGod},'0') as int)>=15) and cast(isnull(#FP{fpPostoji5_14},'0') as int)=1
$cast(isnull(#FP{fpNavrGod},'0') as int)<5 and cast(isnull(#FP{fpPostojiMerenje},'0') as int)=1;
deleteFrom=EHISLica15$ehislica7_14$ehismerenje;
deleteWhere=GOD=d.god and MES=d.mes and pa2Reg=d.pa2Reg and pa3Ops=d.pa3Ops and pa4PopKrug=d.pa4PopKrug and pa5RbrDom=d.pa5RbrDom and RbrClan=d.RbrClan
$GOD=d.god and MES=d.mes and pa2Reg=d.pa2Reg and pa3Ops=d.pa3Ops and pa4PopKrug=d.pa4PopKrug and pa5RbrDom=d.pa5RbrDom and RbrClan=d.RbrClan
$GOD=d.god and MES=d.mes and pa2Reg=d.pa2Reg and pa3Ops=d.pa3Ops and pa4PopKrug=d.pa4PopKrug and pa5RbrDom=d.pa5RbrDom and RbrClan=d.RbrClan;

deleteIF=cast(isnull(#FP{fpNavrGod},'0') as int)<15 and cast(isnull(#FP{fpPostoji15},'0') as int)=1
$(cast(isnull(#FP{fpNavrGod},'0') as int)<5 or cast(isnull(#FP{fpNavrGod},'0') as int)>=15) and
cast(isnull(#FP{fpPostoji5_14},'0') as int)=1
$cast(isnull(#FP{fpNavrGod},'0') as int)<5 and cast(isnull(#FP{fpPostojiMerenje},'0') as int)=1;
stopif=isnull(GodRodj,0)=2019 and isnull(indrodj,0)=2;
stopmsg=Дете није могло да се роди након датума анкетирања.;
}

6) Household structure resets with navigation and assignment (deleteQuestion variant)

#{Ne=;Min=1;Max=7;
StopIf=cast(isnull(Korist,0) as int) in (5,6) and cast(isnull(KoZiviuStanu,0) as int)<>4$
cast(isnull(Korist,0) as int)=4 and cast(isnull(KoZiviuStanu,0) as int)<>4$
cast(isnull(Korist,0) as int)=2 and cast(isnull(KoZiviuStanu,0) as int)=1$
cast(isnull(Korist,0) as int)=2 and cast(isnull(KoZiviuStanu,0) as int)=4$
cast(isnull(Korist,0) as int)=2 and cast(isnull(KoZiviuStanu,0) as int)=5$
cast(isnull(Korist,0) as int)=1 and cast(isnull(KoZiviuStanu,0) as int)=2$
cast(isnull(Korist,0) as int)=1 and cast(isnull(KoZiviuStanu,0) as int)=3$
cast(isnull(Korist,0) as int)=1 and cast(isnull(KoZiviuStanu,0) as int)=4$
cast(isnull(Korist,0) as int)<>1 and cast(isnull(KoZiviuStanu,0) as int)=5$
cast(isnull(Korist,0) as int)=3 and cast(isnull(KoZiviuStanu,0) as int)<>4;
StopMSG=Напуштен стан би требало да буде празан. Проверите унос код питања 4.$
Стан за обављање делатности би требало да буде празан. Проверите унос код питања 4.$
У стану који користе само привремено присутна лица или лица која се не пописују не могу живети једно или више домаћинстава. Проверите унос код питања 4.$Стан који користе само привремено присутна лица или лица која се не пописују не може да буде празан. Проверите унос код питања 4.$Стан који користе само привремено присутна лица или лица која се не пописују се не може користити као други стан домаћинства. Проверите унос код питања 4.$У настањеном стану не могу да живе само привремено присутна лица. Проверите унос код питања 4.$У настањеном стану не могу да живе само лица која се не обухватају пописом. Проверите унос код питања 4.$Настањен стан не може да буде празан. Проверите унос код питања 4.$Ако се стан користи као други стан домаћинства он мора да буде настањен. Проверите унос код питања 4.$Стан који се користи за одмор и сезонске радове би требало да буде празан. Проверите унос код питања 4.;
SkipTo=BrojDom$UBrLicaStan$ButtonNext;SkipIf=cast(isnull(KoZiviuStanu,0) as
int)=1$cast(isnull(KoZiviuStanu,0) as int)=2$cast(isnull(KoZiviuStanu,0) as int) in
(3,4,5,6,7);skippedenabledfalse;skippedsetemptyExcept=BrojDom$BrojDom,UBrLicaStan;skippedse
temptyExceptIf=cast(isnull(KoZiviuStanu,0) as int)=2$cast(isnull(KoZiviuStanu,0) as int) in (3,4,5,6,7);
AssignTo=BrojDom$BrojDom,UBrLicaStan;AssignWhat=0$0;AssignIf=cast(isnull(KoZiviuStanu,0) as
int)=2$cast(isnull(KoZiviuStanu,0) as int) in (3,4,5,6,7);
deleteFrom=P2_Dom,SpisakLica,P1;
deleteIF=cast(isnull(KoZiviuStanu,0) as int)=2 and cast(isnull(#FP{fpCountP2_Dom},0) as int)>0;
deleteWhere=RbrDom>0;
deleteQuestion=true}

7) Roster truncation driven by user count (deleteQuestion=true)

#{Ne=;Min=1;Max=99;
deleteFrom=P2_Dom,SpisakLica,P1;
deleteIF=#FP{fpCountP2_Dom}>BrojDom;
deleteWhere=RbrDom>BrojDom;
deleteQuestion=true}

8) Duplicate of household reset with explicit multi-target assignment

#{Ne=;Min=1;Max=7;
StopIf=cast(isnull(Korist,0) as int) in (5,6) and cast(isnull(KoZiviuStanu,0) as int)<>4$
cast(isnull(Korist,0) as int)=4 and cast(isnull(KoZiviuStanu,0) as int)<>4$
cast(isnull(Korist,0) as int)=2 and cast(isnull(KoZiviuStanu,0) as int)=1$
cast(isnull(Korist,0) as int)=2 and cast(isnull(KoZiviuStanu,0) as int)=4$
cast(isnull(Korist,0) as int)=2 and cast(isnull(KoZiviuStanu,0) as int)=5$
cast(isnull(Korist,0) as int)=1 and cast(isnull(KoZiviuStanu,0) as int)=2$
cast(isnull(Korist,0) as int)=1 and cast(isnull(KoZiviuStanu,0) as int)=3$
cast(isnull(Korist,0) as int)=1 and cast(isnull(KoZiviuStanu,0) as int)=4$
cast(isnull(Korist,0) as int)<>1 and cast(isnull(KoZiviuStanu,0) as int)=5$
cast(isnull(Korist,0) as int)=3 and cast(isnull(KoZiviuStanu,0) as int)<>4;
StopMSG=Напуштен стан би требало да буде празан. Проверите унос код питања 4.$
Стан за обављање делатности би требало да буде празан. Проверите унос код питања 4.$
У стану који користе само привремено присутна лица или лица која се не пописују не могу живети једно или више домаћинстава. Проверите унос код питања 4.$Стан који користе само привремено присутна лица или лица која се не пописују не може да буде празан. Проверите унос код питања 4.$Стан који се користи за одмор и сезонске радове би требало да буде празан. Проверите унос код питања 4.;
SkipTo=BrojDom$UBrLicaStan$ButtonNext;SkipIf=cast(isnull(KoZiviuStanu,0) as int)=1$cast(isnull(KoZiviuStanu,0) as int)=2$cast(isnull(KoZiviuStanu,0) as int) in (3,4,5,6,7);skippedenabledfalse;skippedsetemptyExcept=BrojDom$BrojDom,UBrLicaStan;skippedsetemptyExceptIf=cast(isnull(KoZiviuStanu,0) as int)=2$cast(isnull(KoZiviuStanu,0) as int) in (3,4,5,6,7);
AssignTo=BrojDom$BrojDom$UBrLicaStan;AssignWhat=0$0$0;AssignIf=cast(isnull(KoZiviuStanu,0) as int)=2$cast(isnull(KoZiviuStanu,0) as int) in (3,4,5,6,7)$cast(isnull(KoZiviuStanu,0) as int) in (3,4,5,6,7);
deleteFrom=P2_Dom,SpisakLica,P1;
deleteIF=cast(isnull(KoZiviuStanu,0) as int)=2 and cast(isnull(#FP{fpCountP2_Dom},0) as int)>0;
deleteWhere=RbrDom>0;
deleteQuestion=true}

9) Roster truncation by count (duplicate form)

#{Ne=;Min=1;Max=99;
deleteFrom=P2_Dom,SpisakLica,P1;
deleteIF=#FP{fpCountP2_Dom}>BrojDom;
deleteWhere=RbrDom>BrojDom;
deleteQuestion=true}

Best Practices

• Always combine DeleteFrom + DeleteWhere + DeleteIf to avoid accidental mass deletions.

• Prefer explicit casts when comparing with #FP{...} variables.
• Pair deletes with Assign/Skip to keep UI and counters in sync.
• When truncating rosters/trace tables, delete from most dependent table up to parent to respect any in-memory dependency logic.

Common Errors & Diagnostics
• Missing DeleteWhere → unintended full-table deletes (avoid).
• Type mismatch (#FP text vs numeric) → no-op or runtime error; fix with cast().
• Condition never true → Delete doesn't fire; verify DeleteIf logic and event timing.
• StopIf tripped → Delete won't run; check precedence.

Performance Guidance
• Keep DeleteWhere predicates sargable (simple comparisons, avoid heavy string ops).
• Limit deleteFrom to only necessary tables to minimize rebind overhead.
• Use early validation to prevent unnecessary delete/insert churn.

Security & Integrity
• Deletes are RAM-scoped during data entry; persistence follows standard save pipeline.
• Ensure business rules prevent loss of required evidence rows; consider soft-delete patterns (flags) when needed.
• If using deleteQuestion, ensure UX clarifies impacts to the user.

Testing Checklist
[ ] DeleteIf true path removes only intended rows
[ ] DeleteIf false path performs no deletions
[ ] Type-safe comparisons using cast() where needed
[ ] Dependent grids/labels refresh after delete
[ ] Combined with Skip/Assign, UI remains consistent
[ ] Edge case: zero rows to delete is handled gracefully
[ ] Re-adding rows after delete behaves as expected

Quick Reference (Cheat Sheet)
deleteFrom=Trace1,Trace2,Lica;
deleteWhere=rbr>cast(#FP{fprbr} as integer);
deleteIF=#FP{fprbrTRACE1}>#FP{fprbr};

Cross-References
• Assign - synchronize counters/flags after deletions.
• Skip - navigate away from now-irrelevant sections after cleanup.
• Stop - prevent deletes under invalid states.
• OnSaveNext / OnExit - event-level delete variants.
• DataGridView - verify visual state after delete.

Minimal Metadata Examples

Minimal single-table delete:
#{deleteFrom=Trace1;deleteWhere=rbr>#FP{fprbr};deleteIF=1=1;}

Multiple tables with casting:
#{deleteFrom=Trace1,Trace2;deleteWhere=rbr>cast(#FP{fprbr} as integer);deleteIF=#FP{flagDelete}=1;}

MESSAGE FUNCTIONS

Purpose
Message functions display user-facing dialogs or inline panels to inform, warn, or confirm actions during data entry and validation. They support simple OK-only notifications, OK/Cancel confirmations with default focus on Cancel, and option-style prompts that can branch workflow (via SkipTo/Action). Texts can be inlined (MsgIs/SimpleMsgIs/NoteIs) or centralized in _ISTMessages for localization and reuse.

Where to Define (Metadata Placement)
Use these functions inside the per-control validation block in the _ISTTablesColumns.columnAttributes field or in form/table-level validation rules (_ISTRulesDataValidation.error). Centralized texts referenced by keys (e.g., ISTMSG###) are stored in the _ISTMessages table.

Syntax
Minimal OK-only notification (inline text):
#{
SimpleMsgIs=TEXT;
SimpleMsgIf=CONDITION;
}
OK/Cancel confirmation (inline text):
#{
MsgIs=TEXT;
MsgIf=CONDITION;
}
Message using centralized dictionary (_ISTMessages):
#{
MsgIs=ISTMSG123; // key in _ISTMessages; supports
 for newline
MsgIf=CONDITION;
}

Option-style dialog with branching (multi-option):
#{
OptionMsgIs=OPTION_DEF_1$OPTION_DEF_2;
OptionMsgIf=COND_1$COND_2;
OptionMsgSkipTo=TARGETS_1$TARGETS_2;

OptionMsgAction=ACTIONS_1$ACTIONS_2;
}
Form-note label (writes to a visible NOTE area on the form):
#{
NoteIs=TEXT;
NoteIf=CONDITION;
}

Parameters
SimpleMsgIs / MsgIs / NoteIs:
Literal text to display, or a key from _ISTMessages (e.g., ISTMSG120). '
' renders as a new line.
SimpleMsgIf / MsgIf / NoteIf:
Boolean expression; when true, the message/note appears. Omit to always show.
OptionMsgIs:
Dialog definition(s); parallel lists separated by $ define multiple option prompts.
OptionMsgIf:
Condition(s) that gate each option dialog (parallel with OptionMsgIs).
OptionMsgSkipTo:
Targets to jump to by option result (parallel list).
OptionMsgAction:
Action(s) to perform by option result (parallel list).

Execution Model
• Evaluated on the control's validation event (on leave/commit) or at the rule scope where defined.
• If multiple messages are defined, they are evaluated in declared order. Use StopIf/StopMsg to halt further execution.
• MsgIs shows an OK/Cancel dialog (focus defaults to Cancel). Choosing Cancel returns focus to the current control (or target if MsgFocusOn is used).
• SimpleMsgIs shows an OK-only dialog and continues.
• OptionMsg* opens a selectable options dialog; the chosen option wires to SkipTo/Action entries in the same ordinal position.
• NoteIs writes to a form-level note label (non-modal) and persists until changed/cleared.

Data Types & Formatting
• Inline texts support plain text and '
' markers for new lines.
• Centralized texts in _ISTMessages are recommended for localization and reuse; use keys (ISTMSG###).
• Placeholders/virtual fields (#FP{...}) may be used inside conditions (MsgIf/OptionMsgIf).

Interactions with Other Functions

• Stop*: Use StopIf/StopMsg(or StopMsgOnPnl) around messages to block save or navigation.
• Skip*: Combine OptionMsg* with SkipTo to route the flow by user choice.
• Assign*: Capture outcomes (e.g., set a flag on Cancel) with AssignTo/AssignWhat/AssignIf in the same rule block.
• Visible/Enabled/ReadOnly: Gate the display or effect of messages by UI state.
• Delete/Exec: Confirm destructive actions with MsgIs before DeleteFrom/Exec are triggered.
• OnSaveNext*/OnExit*/OnReturn*: Dedicated variants exist at form events (see cross-references).

Examples

1) Simple OK-only notification

```
#{
SimpleMsgIs='Забелешка: проверите унос.';
SimpleMsgIf=isnull(P3,0)=2;
}
```

2) OK/Cancel confirmation, inline text

```
#{
MsgIs='Да ли желите да наставите?';
MsgIf=cast(isnull(amount,0) as int)>1000;
}
```

3) OK/Cancel using _ISTMessages key

```
#{
MsgIs=ISTMSG105;   // e.g., 'Пажња: унос ће променити постојеће вредности.<br>Наставити?'
MsgIf=isnull(flagUpdate,0)=1;
}
```

4) Option-style dialog that branches (parallel lists)

```
#{
OptionMsgIs='Изаберите радњу:|[1] Сачувај|[2] Одбаци'$'Изаберите наредни корак:|[1] У форми|[2] Грид';
OptionMsgIf=1=1$1=1;
OptionMsgSkipTo=NextStep_Save$GridViewOpen;
OptionMsgAction=SaveRowInGrid$Open=SomeView;
}
```

5) Form note label (persistent)

```
#{
NoteIs='Унос је у току - проверите обавезна поља.';
NoteIf=1=1;
}
```

6) Using message with validation stop

```
#{
StopIf=isnull(PIN,'')='';
StopMsg=ISTMSG050;  // 'ПИН је обавезан.'
MsgIs='Попуните ПИН пре наставка.';
MsgIf=1=1;
}
```

Best Practices

• Centralize user-facing text in _ISTMessages for consistency and localization; keep inline texts short.
• Prefer SimpleMsgIs for informational notes; use MsgIs (OK/Cancel) only when the user's confirmation is required.
• Keep OptionMsg* choices between 2–4 options; beyond that use a dedicated form or grid.
• Always combine potentially destructive actions (Delete/Exec) with a confirmation MsgIs.
• Include actionable next steps in the message body; avoid generic warnings without guidance.

Common Errors & Diagnostics

• Message never appears → Verify MsgIf evaluates to true; check field casing and isnull/cast usage.
• Wrong text shows → Ensure you are pointing to the correct ISTMSG### key. Remember '
' for line breaks.
• Parallel list mismatch (OptionMsg*) → Ensure each OptionMsgIs has a matching OptionMsgIf/SkipTo/Action entry count using '$' separators.
• Modal dead-ends → If MsgIs (Cancel) leads nowhere, add SkipTo or focus management to return the user to a sensible control.

Performance Guidance

• Prefer _ISTMessages keys over large inline literals to keep metadata compact.
• Keep MsgIf expressions simple; offload heavy checks to StopIf/Exec logic or precomputed flags.
• Avoid excessive sequential dialogs; consolidate into a single OptionMsg* where appropriate.

Security & Integrity

• Do not expose sensitive values (IDs, credentials) in message bodies.
• Treat user choices from MsgIs/OptionMsg* as untrusted; apply server-side validation on save.
• Localize all user-facing text via _ISTMessages to prevent ad-hoc edits in production.

Testing Checklist

[ ] MsgIf/OptionMsgIf conditions cover positive/negative cases
[ ] Correct ISTMSG### keys resolve and render '
' as new lines
[ ] Cancel path returns focus to the right control or maintains state
[ ] OptionMsg* paths map 1:1 to SkipTo/Action lists
[ ] Destructive actions require a confirmation dialog
[ ] No double dialogs on a single event unless intentional

Quick Reference (Cheat Sheet)

Inline OK-only: SimpleMsgIs=TEXT; SimpleMsgIf=COND;
Inline OK/Cancel: MsgIs=TEXT_OR_ISTMSG###; MsgIf=COND;
Options dialog: OptionMsgIs=A$B; OptionMsgIf=C$D; OptionMsgSkipTo=T1$T2; OptionMsgAction=X1$X2;
Form note: NoteIs=TEXT; NoteIf=COND;

Cross-References

• Stop - hard validation and message panels
• Skip - routing after messages and options

• Assign - capturing user choices and flags
• Delete - guard destructive operations with confirmations
• Exec - confirm long-running or external operations
• OnSaveNext / OnExit / OnReturn - event-scoped message variants

Minimal Metadata Examples

1) Minimal OK/Cancel message

#{
MsgIs='Наставити?';
MsgIf=1=1;
}

2) Multi-message (parallel lists)

#{
OptionMsgIs='Избор 1|[1] Да|[2] Не'$'Избор 2|[1] Отвори|[2] Откажи';
OptionMsgIf=1=1$cast(isnull(flag,0) as int)=1;
OptionMsgSkipTo=Next$Cancel;
OptionMsgAction=Open=SomeView$CancelEditRowInGrid;
}

VISIBLE (VISIBLETRUE / VISIBLEFALSE)

Purpose

Control runtime visibility of UI elements based on declarative rules. Use to progressively disclose fields, reduce clutter, and guide operators.

Where to Define (Metadata Placement)

- _ISTTablesColumns.columnAttributes - per-field visibility rules.
- _ISTRulesDataValidation.validationEvent - conditional visibility tied to validation flow.
- Panel/Group level metadata - to toggle containers that hold multiple controls.

Syntax

#{
VisibleTrue=FieldA,FieldB,PanelX;
VisibleTrueIf=cast(isnull(Status,0) as int)=1;
}
#{
VisibleFalse=FieldC$PanelY;
VisibleFalseIf=isnull(FlagHide,0)=1$cast(isnull(Type,0) as int) IN (3,4);
}

Parameters

- VisibleTrue / VisibleFalse - comma-separated list of target controls or panels.
- VisibleTrueIf / VisibleFalseIf - expression(s); use $ to pair with multiple targets in order.
- Expressions support: d.field, #FP{param}, isnull(), cast(), IN(), LIKE, etc.

Execution Model

- Evaluated when form opens, on field leave, and whenever referenced variables change.
- Multiple directives can coexist; last one in evaluation order wins per control.
- Hidden controls do not receive focus or keystrokes; values are preserved unless Skip/Delete clears them.

Data Types & Formatting

- Conditions must return boolean (True/False).
- String constants in quotes; numeric using cast() when necessary.
- Use case-insensitive control names as defined in metadata.

Interactions with Other Functions

- Pairs well with **Enable** and **ReadOnly** for staged workflows.
- Combine with **Skip** to jump past hidden sections.
- Use **Color** to highlight newly shown fields.

Examples

```
#{
VisibleTrue=ReasonText;
VisibleTrueIf=cast(isnull(HasReason,0) as int)=1;
}
#{
VisibleFalse=OptionalGroup$SidePanel;
VisibleFalseIf=isnull(HideOptional,0)=1$isnull(Mode,'')='COMPACT';
}
```

Best Practices

- Prefer VisibleFalse for safety (start visible unless explicitly hidden).
- Keep conditions simple; complex logic -> compute a helper flag first via Assign.
- Avoid flicker: tie visibility to stable fields, not rapidly changing inputs.

Common Errors & Diagnostics

- Condition never true: verify token names and data types.
- Control not found: ensure target exists in ISTPolja and is on the current form.
- Conflicting rules: document precedence or consolidate into one directive.

Performance Guidance

- Minimize expensive functions in conditions.
- Group related controls under a Panel and toggle the panel instead.

Security & Integrity

- Visibility does not equal authorization - server-side validation must still enforce rules.

Testing Checklist

- [] Controls appear/disappear under all expected states.

- [] Hidden controls do not receive focus.
- [] No conflicting VisibleTrue/False for same control in same state.

Quick Reference (Cheat Sheet)
VisibleTrue=CTRL1,CTRL2; VisibleTrueIf=COND
VisibleFalse=CTRL; VisibleFalseIf=COND // $ to pair multiple

Cross-References
Assign; Stop; Skip; Delete; Message; Visible; ReadOnly; Exec; Enable; Color; OnSaveNext; OnExit; AfterUpdateQC; OnReturn; MultiResponseTextBox; DataGridView; AutoCompleteTextBox; Relacija/WebRelacija.

Minimal Metadata Examples
#{
VisibleTrue=Extra1,Extra2; VisibleTrueIf=cast(isnull(tip,0) as int)=2;
}

READONLY

Purpose
Lock one or more controls from user editing while still showing current values.

Where to Define (Metadata Placement)
- _ISTTablesColumns.columnAttributes - per-field read-only rules.
- _ISTRulesDataValidation.validationEvent - dynamic, condition-based locking.

Syntax
#{
ReadOnly=Field1,Field2;
ReadOnlyIf=isnull(IsFinal,0)=1;
}
#{
ReadOnly=FieldA$FieldB;
ReadOnlyIf=isnull(Stage,0)>=3$isnull(Stage,0)>=2;
}

Parameters
- ReadOnly - comma list of targets.
- ReadOnlyIf - single or $-paired conditions.

Execution Model
- Evaluated on load and on dependent changes.
- ReadOnly controls can still be copied, validated and assigned by system logic.

Data Types & Formatting

Same as Visible conditions.

Interactions with Other Functions

- Combine with **EnableFalse** to prevent focus and tab-stop if needed.
- Use **Color** to visually indicate locked content.

Examples

#{
ReadOnly=MBR,OPS;
ReadOnlyIf=cast(isnull(IsImported,0) as int)=1;
}

Best Practices

- Prefer ReadOnly for audit trails; avoid silently changing values afterward.
- If a field must remain interactive (e.g., for selection), use EnableFalse instead.

Common Errors & Diagnostics

- User still edits: Ensure the control is not duplicated or re-enabled elsewhere.
- Dependent logic fails: account for read-only values in calculations.

Performance Guidance

Negligible impact.

Security & Integrity

Locking is UI-side; enforce with server validation where critical.

Testing Checklist

- [] Field shows current value but is not editable.
- [] Tab order skips or stops as intended.

Quick Reference (Cheat Sheet)

ReadOnly=CTRL1,CTRL2; ReadOnlyIf=COND

Cross-References

Assign; Stop; Skip; Delete; Message; Visible; ReadOnly; Exec; Enable; Color; OnSaveNext; OnExit; AfterUpdateQC; OnReturn; MultiResponseTextBox; DataGridView; AutoCompleteTextBox; Relacija/WebRelacija.

Minimal Metadata Examples

#{
ReadOnly=AllTotals; ReadOnlyIf=cast(isnull(Frozen,0) as int)=1;
}

*EXEC (Exec / On*Exec)*

Purpose

Execute stored procedures or SQL statements from declarative rules (validation events, Save/Exit hooks, etc.).

Where to Define (Metadata Placement)

- _ISTRulesDataValidation.validationEvent - general Exec triggers.
- OnSaveNextExec / AfterUpdateQCExec / OnReturnExec - context-specific event hooks.

Syntax

#{
Exec=EXEC usp_Process d.god, d.mbops, #FP{fpUserId};
}
#{
OnSaveNextExec=EXEC usp_sync d.god, ISTQC$EXEC usp_log #FP{fpUserId};
}

Parameters

- Exec / On*Exec - one or more statements separated by $; evaluated left-to-right.
- Parameters: d.field, #FP{...}, constants, ISTQC (if applicable).

Execution Model

- Runs after validation (unless blocked by StopIf).
- Chained execs stop on first failure (recommended).

Data Types & Formatting

- Use EXEC proc param1, param2 format.
- Cast/convert as required by stored procedure signature.

Interactions with Other Functions

- Often paired with **OnSaveNext**/**AfterUpdateQC**.
- Use **Message** for user feedback after Exec.

Examples

#{
Exec=EXEC usp_RecalcTotals d.god, d.ops;
}
#{
OnSaveNextExec=EXEC usp_CommitLine d.pk$EXEC usp_LogAction #FP{fpUser};
}

Best Practices

- Keep procedures idempotent; events might fire more than once.
- Log failures server-side and surface concise messages to users.

Common Errors & Diagnostics

- Parameter mismatch: verify order and data types.
- Silent failure: ensure the engine is configured to bubble up errors.

Performance Guidance
- Batch heavy work; avoid long-running execs on keystroke events.

Security & Integrity
- Use parametrized SPs; avoid concatenating user input into SQL.

Testing Checklist
- [] Verify parameters are bound correctly.
- [] Simulate failure path - does UI report gracefully?

Quick Reference (Cheat Sheet)
Exec=EXEC proc d.f1, #FP{p}; OnSaveNextExec=EXEC proc ISTQC$EXEC proc2

Cross-References
Assign; Stop; Skip; Delete; Message; Visible; ReadOnly; Exec; Enable; Color; OnSaveNext; OnExit; AfterUpdateQC; OnReturn; MultiResponseTextBox; DataGridView; AutoCompleteTextBox; Relacija/WebRelacija.

Minimal Metadata Examples
#{
Exec=EXEC usp_touch d.id;
}

ENABLE (ENABLETRUE / ENABLEFALSE)

Purpose
Enable/disable controls dynamically. Disabled controls are visible but not focusable or editable.

Where to Define (Metadata Placement)
- _ISTTablesColumns.columnAttributes
- _ISTRulesDataValidation.validationEvent

Syntax
#{
EnableTrue=Field1,Field2;
EnableTrueIf=cast(isnull(Stage,0) as int)>=2;
}
#{
EnableFalse=GroupA$Notes;
EnableFalseIf=isnull(LockAll,0)=1$isnull(HasNotes,0)=0;
}

Parameters

- EnableTrue / EnableFalse - targets; comma list.
- EnableTrueIf / EnableFalseIf - boolean expressions; parallel list with $ pairing optional.

Execution Model

- Evaluated on form load and on dependency change.
- If both True and False apply, last evaluated wins.

Data Types & Formatting

Same as Visible/ReadOnly conditions.

Interactions with Other Functions

- Combine with **ReadOnly** for nuanced UX.
- Use **Visible** to hide entirely when disabled is not enough.

Examples

#{
EnableFalse=AllInputs;
EnableFalseIf=cast(isnull(IsSubmitted,0) as int)=1;
}

Best Practices

- Disable in error states only as needed; otherwise inform via Message and keep fields active for correction.

Common Errors & Diagnostics

- Control remains enabled: conflicting rule elsewhere - consolidate conditions.

Performance Guidance

Negligible.

Security & Integrity

As with Visible/ReadOnly, enforce also on server.

Testing Checklist

- [] Disabled controls are not focusable.
- [] Re-enable when prerequisites satisfied.

Quick Reference (Cheat Sheet)

EnableTrue=CTRLs; EnableTrueIf=COND EnableFalse=CTRLs; EnableFalseIf=COND

Cross-References

Assign; Stop; Skip; Delete; Message; Visible; ReadOnly; Exec; Enable; Color; OnSaveNext; OnExit; AfterUpdateQC; OnReturn; MultiResponseTextBox; DataGridView; AutoCompleteTextBox; Relacija/WebRelacija.

Minimal Metadata Examples

#{
EnableTrue=ExtraBox; EnableTrueIf=isnull(flag,0)=1;
}

COLOR (ColorTo / ColorWhat / ColorIf)

Purpose

Apply color styling to controls to provide visual cues (e.g., errors, required state, stage highlighting).

Where to Define (Metadata Placement)

- _ISTTablesColumns.columnAttributes - per-field cues.
- _ISTRulesDataValidation.validationEvent - dynamic cues tied to workflow.

Syntax

#{
ColorTo=Field1,Field2;
ColorWhat=BackColor=LightYellow,ForeColor=Black;
ColorIf=cast(isnull(IsRequired,0) as int)=1;
}
#{
ColorTo=Total;
ColorWhat=BackColor=Tomato;
ColorIf=cast(isnull(Total,0) as int)<0;
}

Parameters

- ColorTo - targets (comma list).
- ColorWhat - one or more key=value pairs for ForeColor/BackColor; comma-separated.
- ColorIf - condition; $-paired list supported per target if needed.

Execution Model

- Re-evaluated on load and when dependencies change.
- Last matching rule wins per property.

Data Types & Formatting

- Color names per System.Drawing.KnownColor (or #RRGGBB if supported).

Interactions with Other Functions

- Use with **Stop/Message** to emphasize blocking fields.
- Combine with **Visible/Enable/ReadOnly** for comprehensive UX.

Examples

#{
ColorTo=Status;
ColorWhat=BackColor=LightGreen;
ColorIf=cast(isnull(Status,0) as int)=1;
}

Best Practices

- Maintain accessible contrast ratios.
- Be consistent across the project (define a small palette).

Common Errors & Diagnostics

- Unknown color name: fix typo or use a hex value.

Performance Guidance

Minimal impact.

Security & Integrity

Styling only; no data effects.

Testing Checklist

- [] Color applied under expected conditions.
- [] Resets when condition no longer holds.

Quick Reference (Cheat Sheet)

ColorTo=CTRLs; ColorWhat=BackColor=Name[,ForeColor=Name]; ColorIf=COND

Cross-References

Assign; Stop; Skip; Delete; Message; Visible; ReadOnly; Exec; Enable; Color; OnSaveNext; OnExit; AfterUpdateQC; OnReturn; MultiResponseTextBox; DataGridView; AutoCompleteTextBox; Relacija/WebRelacija.

Minimal Metadata Examples

#{
ColorTo=Notes; ColorWhat=BackColor=Khaki; ColorIf=isnull(NeedsReview,0)=1;
}

OnSaveNext (Form Events)*

Purpose

Run actions when the user presses SaveNext (save current record and immediately prepare a new entry).

Where to Define (Metadata Placement)

- _ISTRulesDataValidation.validationEvent (form/panel scope).
- Child forms: OnSaveNextDontClose to keep form open.

Syntax

```
#{
OnSaveNextDeleteFrom=Trace1, Lica;
OnSaveNextDeleteIf=U010110='2';
}
#{
OnSaveNextFocusOn=PKField;
OnSaveNextFocusIf=isnull(NeedFocus,0)=1;
OnSaveNextDontClose;
}
#{
OnSaveNextCloseIf=cast(isnull(CanClose,0) as int)=1;
}
#{
OnSaveNextSimpleMsgIs='Saved.';
OnSaveNextMsgIf=2=2;
}
#{
OnSaveNextMsgIs='Proceed with next?';
OnSaveNextMsgIf=isnull(ConfirmNext,0)=1;
}
#{
OnSaveNextStopIf=cast(isnull(HasErrors,0) as int)=1;
OnSaveNextStopMsg='Correct highlighted fields before continuing.';
OnSaveNextStopMsgFocusOn=ErrField;
}
#{
OnSaveNextStopMsgOnPnl=PNL_SaveHints;
OnSaveNextStopIf=isnull(ShowHints,0)=1;
}
#{
OnSaveNextStopLoopFromGridIf=isnull(StopLoop,0)=1;
}
#{
OnSaveNextExec=EXEC usp_upd_Kontrola_teren d.god,d.mbr;
}
```

Parameters

- DeleteFrom, DeleteIf - RAM tables cleanup after save.
- FocusOn, FocusIf - set focus (child forms; requires DontClose).
- DontClose / CloseIf - lifecycle control for child forms.
- SimpleMsgIs / MsgIs + MsgIf - informational or confirm prompts.
- StopIf / StopMsg (/FocusOn|OnPnl) - block and explain.
- StopLoopFromGridIf - halt DataGridView auto-looping.

- OnSaveNextExec - run stored procedures.

Execution Model
- Order: Stop* → Delete* → Exec → Msg* → Focus/Close behaviors.
- If StopIf triggers, subsequent actions are skipped.

Data Types & Formatting
Conditions mirror standard expression language; Exec uses T-SQL EXEC syntax.

Interactions with Other Functions
- Often combined with **Assign/Skip** to prepare the next record.
- Use **Color/Visible/Enable** to signal what to fix when blocked.

Examples
#{
OnSaveNextDeleteFrom=Trace1,Trace2;
OnSaveNextDeleteIf=cast(isnull(ResetTrace,0) as int)=1;
OnSaveNextMsgIs='Saved successfully. Continue?';
OnSaveNextMsgIf=2=2;
}

Best Practices
- Keep Stop reasons precise; users are in a rapid entry flow.
- Exec calls should be fast; defer heavy jobs to background processing.

Common Errors & Diagnostics
- FocusOn ignored: missing DontClose on child forms.
- DeleteFrom runs unexpectedly: verify DeleteIf logic.

Performance Guidance
Prefer lightweight validation and logging on this path.

Security & Integrity
Deletion affects RAM only; the saved record persists. Server-side checks still apply.

Testing Checklist
- [] StopIf blocks when expected with clear message.
- [] DeleteFrom prunes RAM tables correctly.
- [] Exec parameters correct; messages appear in right order.

Quick Reference (Cheat Sheet)
OnSaveNext[DeleteFrom/If|FocusOn/If|DontClose|CloseIf|SimpleMsgIs|MsgIs/MsgIf|StopIf/StopMsg(/FocusOn|OnPnl)|StopLoopFromGridIf|Exec]

Cross-References

Assign; Stop; Skip; Delete; Message; Visible; ReadOnly; Exec; Enable; Color; OnSaveNext; OnExit; AfterUpdateQC; OnReturn; MultiResponseTextBox; DataGridView; AutoCompleteTextBox; Relacija/WebRelacija.

Minimal Metadata Examples

#{
OnSaveNextSimpleMsgIs='Saved'; OnSaveNextMsgIf=2=2;
}

OnExit (Form Events)*

Purpose

Run validation, messaging, or confirmation logic when the user presses Exit.

Where to Define (Metadata Placement)

- _ISTRulesDataValidation.validationEvent at panel or form scope.

Syntax

#{
OnExitStopIf=isnull(Key,'')='';
OnExitStopMsgIs='Key is mandatory.';
OnExitStopMsgFocusOn=Key;
}
#{
OnExitSimpleMsgIs='Goodbye!'; OnExitMsgIf=2=2;
}
#{
OnExitMsgIs='Exit without saving?'; OnExitMsgIf=cast(isnull(IsDirty,0) as int)=1;
}
#{
OnExitStopMsgOnPnl=PNL_ExitHelp; OnExitStopIf=isnull(ShowPanel,0)=1;
}

Parameters

- OnExitStopIf / OnExitStopMsgIs (/FocusOn|OnPnl) - hard block with explanation.
- OnExitSimpleMsgIs / OnExitMsgIs + OnExitMsgIf - informational or confirm dialog.

Execution Model

- First matching StopIf blocks Exit; otherwise messages/confirmations run.
- FocusOn moves caret to target control after a block.

Data Types & Formatting

Same expression language as other directives; messages are plain text with optional tokens.

Interactions with Other Functions

- Coordinate with **Stop** rules earlier to avoid duplicate messages.

- Pair with **Visible/Enable/ReadOnly** to prepare corrective action.

Examples

#{
OnExitMsgIs='Unsaved changes. Exit?'; OnExitMsgIf=cast(isnull(IsDirty,0) as int)=1;
}

Best Practices

- Keep messages short and actionable.
- Use a single blocking rule where possible to avoid confusion.

Common Errors & Diagnostics

- No effect: confirm that Exit event is wired for the form.
- Focus target invisible/disabled: choose a valid control.

Performance Guidance

Trivial.

Security & Integrity

No persistence; UI flow only.

Testing Checklist

- [] StopIf triggers when expected and prevents Exit.
- [] Confirmation appears only under the intended condition.

Quick Reference (Cheat Sheet)

OnExit[StopIf/StopMsgIs(/FocusOn|OnPnl)|SimpleMsgIs/MsgIs+MsgIf]

Cross-References

Assign; Stop; Skip; Delete; Message; Visible; ReadOnly; Exec; Enable; Color; OnSaveNext; OnExit; AfterUpdateQC; OnReturn; MultiResponseTextBox; DataGridView; AutoCompleteTextBox; Relacija/WebRelacija.

Minimal Metadata Examples

#{
OnExitSimpleMsgIs='Bye'; OnExitMsgIf=2=2;
}

*AfterUpdateQC**

Purpose

React immediately after the Quality Control (ISTQC) flag is updated to 0 or 1: show messages and/or run server procedures.

Where to Define (Metadata Placement)

- _ISTRulesDataValidation.validationEvent near QC workflow.

Syntax

#{AfterUpdateQCOn0IstMsg=QC0_INFO;}
#{AfterUpdateQCOn1IstMsg=QC1_INFO;}
#{AfterUpdateQCExec=EXEC usp_QC_PostUpdate d.god, d.ops, ISTQC;}

Parameters

- AfterUpdateQCOn0IstMsg - message key(s) from ISTMessages when QC becomes 0.
- AfterUpdateQCOn1IstMsg - message key(s) from ISTMessages when QC becomes 1.
- AfterUpdateQCExec - one or more EXEC statements; pass ISTQC as parameter if needed.

Execution Model

- Fires right after QC state flips; order: messages, then Exec.
- Exec should be idempotent; QC may toggle repeatedly.

Data Types & Formatting

Messages are lookup keys; Exec uses standard T-SQL EXEC with d.*, #FP{}, ISTQC parameters.

Interactions with Other Functions

- Combine with **Stop** in QC transition logic to prevent invalid flips.
- OnSaveNext/OnExit can complement with navigation and messaging.

Examples

#{
AfterUpdateQCOn0IstMsg=QC_REJECTED;
AfterUpdateQCOn1IstMsg=QC_PASSED;
AfterUpdateQCExec=EXEC usp_QC_Log d.god, d.mbops, ISTQC;
}
#{
AfterUpdateQCExec=Exec usp_SkloniDatumZakazivanja d.mbops, ISTQC;
}

Best Practices

- Prefer ISTMessages keys for localization.
- Log state changes with timestamps and operator IDs.

Common Errors & Diagnostics

- Message not found: ensure ISTMessages contains the key.
- Exec fails: verify proc signature and parameter order/types.

Performance Guidance

Keep as light as possible; heavy processing should be queued.

Security & Integrity

All server changes must be authorized/validated in the stored procedure.

Testing Checklist

- [] Verify both transitions (to 0 and to 1).
- [] Confirm messages display and Exec receives correct ISTQC.

Quick Reference (Cheat Sheet)
AfterUpdateQCOn0IstMsg=KEYS AfterUpdateQCOn1IstMsg=KEYS AfterUpdateQCExec=EXEC proc ..., ISTQC

Cross-References
Assign; Stop; Skip; Delete; Message; Visible; ReadOnly; Exec; Enable; Color; OnSaveNext; OnExit; AfterUpdateQC; OnReturn; MultiResponseTextBox; DataGridView; AutoCompleteTextBox; Relacija/WebRelacija.

Minimal Metadata Examples
#{
AfterUpdateQCOn1IstMsg=QC1_OK;
AfterUpdateQCExec=EXEC usp_QC_Audit d.god, d.ops, ISTQC;
}

OnReturn Events - Parent Form Actions when Returning from Child Form

Purpose
Provide parent-form side-effects immediately after a child form closes (return). Use to toggle UI, move focus, assign values returned from child, display messages, or block continuation with STOPIF.

Syntax
#{
OnReturnVisibleFalse=ctrl1,ctrl2; OnReturnVisibleFalseIf={COND};
OnReturnVisibleTrue=ctrl1,ctrl2; OnReturnVisibleTrueIf={COND};
OnReturnEnabledFalse=ctrl1,ctrl2; OnReturnEnabledFalseIf={COND};
OnReturnEnabledTrue=ctrl1,ctrl2; OnReturnEnabledTrueIf={COND};
OnReturnFocusOn=ControlName; OnReturnFocusIf={COND};
OnReturnAssignTo=F1,F2; OnReturnAssignWhat=Expr1,Expr2; OnReturnAssignIf={COND};
OnReturnSimpleMsgIs='text'; OnReturnMsgIf={COND};
OnReturnMsgIs='text'; OnReturnMsgIf={COND}; // OK/Cancel, focus on Cancel
}

Parameters
- OnReturnVisibleFalse/True: Comma-separated control names.
- OnReturnEnabledFalse/True: Comma-separated control names.
- OnReturnFocusOn: Single control to receive focus.
- OnReturnAssignTo/What: Parallel lists; use $ to branch per-condition block if needed.
- ...If: Boolean condition determining whether the action fires.

Where to Define (Metadata Placement)

Define in _ISTRulesDataValidation.validationEvent for per-control rules, and/or in _ISTTablesColumns.columnAttributes for control-level properties. Form-level events (OnReturn*, OnSaveNext*, etc.) live in _ISTRulesFormEvents.

Execution Model

Evaluated by the .NET interpreter on control enter/leave and on form events. All conditions (...If) are boolean expressions against current RAM dataset (d.*) and virtual fields (#FP{...}). SQL fragments run against the active connection and respect the current ISTTimePoint (year/month) when applicable.

Data Types & Formatting

- Use cast/convert in expressions where numeric comparison is required.
- Multi-response values are comma-separated tokens (e.g., '1,3,5').
- String constants must be quoted with single quotes.

Interactions with Other Functions

- STOPIF/STOPMSG can short-circuit event flows before actions execute.
- ASSIGN/FOCUS/Visible*/Enabled*/ReadOnly* often used together post-validate.
- EXEC can be chained with grid actions or form-level events.

Best Practices

- Keep conditions explicit and side-effect free (no implicit conversions).
- Prefer field-safe guards (isnull(...)) to avoid null reference errors.
- Log critical event results with MsgIs to aid troubleshooting.

Common Errors & Diagnostics

- Condition evaluates but targets are missing from screen (check metadata wiring).
- Ambiguous control names - ensure unique per form.
- SQL used in actions lacks required parameters (GGG/MMM, etc.).

Performance Guidance

- Cache small reference tables in RAM; avoid per-keystroke SQL calls.
- Keep grids lightweight (limit columns/rows) to speed redraw.
- Avoid excessive DOM toggling (Visible/Enabled) in a single event.

Security & Integrity

- Never embed user input directly into SQL - use parameterization patterns.
- Restrict EXEC targets to vetted stored procedures.
- Use Delete* only against RAM snapshots unless explicitly intended.

Testing Checklist

- [ ] Happy-path run with valid inputs
- [ ] Boundary values (min/max, empty, multi-response counts)
- [ ] Negative cases fire STOPIF/MSG as designed

- [ ] Focus & visibility shift as expected
- [ ] All assignments persist after Save/SaveNext
- [ ] Grid link actions perform and rollback cleanly

Quick Reference (Cheat Sheet)
- Condition fields: d.Field, #FP{Virtual}, constants '...'
- Multi-branch: separate parallel lists with $
- Skips/Deletes: pair targets and conditions 1:1
- Messages: SimpleMsgIs (OK) vs MsgIs (OK/Cancel)

Cross-References
- See: Assign, Stop, Skip, Visible, Enabled, ReadOnly, Exec, Delete, Message
- See controls: AutoCompleteTextBox, ComboBox, DataGridView, AlwaysVisibleLabel

Examples

```
#{
OnReturnVisibleFalse=panelExtra; OnReturnVisibleFalseIf=isnull(d.countRows,0)=0;
OnReturnEnabledTrue=SaveButton; OnReturnEnabledTrueIf=isnull(d.isValid,0)=1;
OnReturnAssignTo=Total; OnReturnAssignWhat=cast(isnull(d.sumChild,0) as int);
OnReturnSimpleMsgIs='Child updates applied.'; OnReturnMsgIf=isnull(d.rows,0)>0;
}
```

Minimal Metadata Examples

```
#{
OnReturnFocusOn=MainField; OnReturnFocusIf=1=1;
}
```

MultiResponse TextBox - Tokenized Input with Count Limits

Purpose

Capture multiple coded answers in a single text box. Enforce per-token type, overall min/max value, and the allowed number of responses.

Syntax

```
#{
Yes=MultiResponseNumeric|MultiResponseAlpha;
Min={MIN}; Max={MAX};
MnResponseCount={MIN_COUNT}; MxResponseCount={MAX_COUNT};
No={DISALLOWED_CODES};
}
```

Parameters
- Yes=MultiResponseNumeric|MultiResponseAlpha: Token type enforcement.
- Min/Max: Allowed token bounds (numeric mode).
- MnResponseCount/MxResponseCount: Required and maximum token count.

- No: Comma-separated disallowed tokens.

Where to Define (Metadata Placement)
Define in _ISTRulesDataValidation.validationEvent for per-control rules, and/or in _ISTTablesColumns.columnAttributes for control-level properties. Form-level events (OnReturn*, OnSaveNext*, etc.) live in _ISTRulesFormEvents.

Execution Model
Evaluated by the .NET interpreter on control enter/leave and on form events. All conditions (...If) are boolean expressions against current RAM dataset (d.*) and virtual fields (#FP{...}). SQL fragments run against the active connection and respect the current ISTTimePoint (year/month) when applicable.

Data Types & Formatting
- Use cast/convert in expressions where numeric comparison is required.
- Multi-response values are comma-separated tokens (e.g., '1,3,5').
- String constants must be quoted with single quotes.

Interactions with Other Functions
- STOPIF/STOPMSG can short-circuit event flows before actions execute.
- ASSIGN/FOCUS/Visible*/Enabled*/ReadOnly* often used together post-validate.
- EXEC can be chained with grid actions or form-level events.

Best Practices
- Keep conditions explicit and side-effect free (no implicit conversions).
- Prefer field-safe guards (isnull(...)) to avoid null reference errors.
- Log critical event results with MsgIs to aid troubleshooting.

Common Errors & Diagnostics
- Condition evaluates but targets are missing from screen (check metadata wiring).
- Ambiguous control names - ensure unique per form.
- SQL used in actions lacks required parameters (GGG/MMM, etc.).

Performance Guidance
- Cache small reference tables in RAM; avoid per-keystroke SQL calls.
- Keep grids lightweight (limit columns/rows) to speed redraw.
- Avoid excessive DOM toggling (Visible/Enabled) in a single event.

Security & Integrity
- Never embed user input directly into SQL - use parameterization patterns.
- Restrict EXEC targets to vetted stored procedures.
- Use Delete* only against RAM snapshots unless explicitly intended.

Testing Checklist
- [ ] Happy-path run with valid inputs
- [ ] Boundary values (min/max, empty, multi-response counts)

- [ ] Negative cases fire STOPIF/MSG as designed
- [ ] Focus & visibility shift as expected
- [ ] All assignments persist after Save/SaveNext
- [ ] Grid link actions perform and rollback cleanly

Quick Reference (Cheat Sheet)

- Condition fields: d.Field, #FP{Virtual}, constants '...'
- Multi-branch: separate parallel lists with $
- Skips/Deletes: pair targets and conditions 1:1
- Messages: SimpleMsgIs (OK) vs MsgIs (OK/Cancel)

Cross-References

- See: Assign, Stop, Skip, Visible, Enabled, ReadOnly, Exec, Delete, Message
- See controls: AutoCompleteTextBox, ComboBox, DataGridView, AlwaysVisibleLabel

Examples

#{MultiResponseAlpha;MxResponseCount=3;}
#{min=0;max=10;Yes=MultiResponseNumeric;MxResponseCount=7;MnResponseCount=3}
#{MultiResponseNumeric;MxResponseCount=4;No=9,7,6;}
#{No=;Yes=MultiResponseNumeric;Min=1;Max=11;MxResponseCount=5;MnResponseCount=5;}
#{No=;Yes=MultiResponseNumeric;Min=1;Max=12;SkipTo=p_7_multi$p_6_4_1$p_6_4_2;SkipIf=isnull(p_6_multi,'') not like '%6%' and isnull(p_6_multi,'') not like '%4%'$isnull(p_6_multi,'') like '%6%'$isnull(p_6_multi,'') like '%4%';skippedenabledfalse;SkippedSetEmpty}
#{SkipTo=p_7_multi$p_6_4_2;SkipIf=isnull(p_6_multi,'') not like '%4%' $isnull(p_6_multi,'') like '%4%';StopMsg=Obavezan unos!;StopIf=(len(p_6_4_1)=0 or p_6_4_1='');skippedsetempty;skippedenabledfalse;}
#{No=;Yes=MultiResponseNumeric;Min=1;Max=11;MnResponseCount=5;MxResponseCount=5;SkipTo=p_8_1$p_7_8_other;SkipIf=isnull(p_7_multi,'') not like '%8%'$isnull(p_7_multi,'') like '%8%';skippedenabledfalse;SkippedSetEmpty}
#{No=;Yes=MultiResponseNumeric;Min=1;Max=9;MnResponseCount=5;MxResponseCount=5;}
#{No=;Yes=MultiResponseNumeric;Min=1;Max=13;SkipTo=p_7_multi$p_6_2_1$p6_dodatno_7;SkipIf=isnull(p_6_multi,'') not like '%6%' and isnull(p_6_multi,'') not like '%4%'$isnull(p_6_multi,'') like '%4%'$isnull(p_6_multi,'') like '%6%';skippedenabledfalse;SkippedSetEmpty}
#{SkipTo=p_7_multi$p6_dodatno_7;SkipIf=isnull(p_6_multi,'') not like '%6%' $isnull(p_6_multi,'') like '%6%';StopMsg=Obavezan unos!;StopIf=(len(p_6_2_1)=0 or p_6_2_1='');skippedenabledfalse;skippedsetempty;}
#{No=;Yes=MultiResponseNumeric;MxResponseCount=5;MnResponseCount=5;Min=1;Max=10;SkipTo=p_8_1$p_7_10_drugo;SkipIf=isnull(p_7_multi,'') not like '%10%'$isnull(p_7_multi,'') like '%10%';skippedenabledfalse;skippedsetempty}

Minimal Metadata Examples

#{Yes=MultiResponseNumeric; MnResponseCount=1; MxResponseCount=3;}

DATAGRIDVIEW - READ-ONLY TABULAR BROWSER WITH COMMAND COLUMNS

Purpose

Display and interact with tabular datasets. By default, grid is read-only, supports links, inline row editing (optional), and custom command columns. Often used for code lists or child rows. If parameter Rooster is used, then grid is editable (readOnly=false)

Syntax

```
#{
// Core data
GridSource={TABLE} | GridSourceView={VIEW} | GridDataSource=(SELECT ...);
GridSourceWhere={WHERE}; GridOrderBy={ORDER BY};
GridColumns=Header=Col,Header2=Col2; GridColumnsWidth=120,80;

// Command columns
GridLinkText=Edit$Delete$Open;
GridLinkAction=Edit$Delete=TableOrView$Open=TableOrView;
GridLinkWidth=60$60$60;
GridLinkStopIf={COND1}${COND2}${COND3};
GridLinkStopIfMsg='Text1'$'Text2'$'Text3';

// Column-as-link
GridColumnLink=code$docNo;
GridColumnLinkAction=Open=Docs$Exec;

// Optional adds
GridAddColumn=Title=RAM.Code$Type=RAM.Type;
GridAddColumnOnPosition=1$3;

// Sizing & style
GridHeight=400; GridWidth=800; GridRowsHeight=22; GridColumnsAutoResize;
GridTitle='Available Items'; GridTitleFontBold; GridTitleFontSize=10;
}
```

Parameters

- GridSource / GridSourceView / GridDataSource: pick exactly one data source.
- GridColumns / GridColumnsWidth: headers and widths in order.
- GridLinkText / GridLinkAction: parallel $-lists for command columns.
- GridColumnLink / GridColumnLinkAction: make a data column clickable.

Where to Define (Metadata Placement)

Define in _ISTRulesDataValidation.validationEvent for per-control rules, and/or in _ISTTablesColumns.columnAttributes for control-level properties. Form-level events (OnReturn*, OnSaveNext*, etc.) live in _ISTRulesFormEvents.

Execution Model

Evaluated by the .NET interpreter on control enter/leave and on form events. All conditions (...If) are boolean expressions against current RAM dataset (d.*) and virtual fields (#FP{...}). SQL fragments run against the active connection and respect the current ISTTimePoint (year/month) when applicable.

Data Types & Formatting

- Use cast/convert in expressions where numeric comparison is required.
- Multi-response values are comma-separated tokens (e.g., '1,3,5').
- String constants must be quoted with single quotes.

Interactions with Other Functions

- STOPIF/STOPMSG can short-circuit event flows before actions execute.
- ASSIGN/FOCUS/Visible*/Enabled*/ReadOnly* often used together post-validate.
- EXEC can be chained with grid actions or form-level events.

Best Practices

- Keep conditions explicit and side-effect free (no implicit conversions).
- Prefer field-safe guards (isnull(...)) to avoid null reference errors.
- Log critical event results with MsgIs to aid troubleshooting.

Common Errors & Diagnostics

- Condition evaluates but targets are missing from screen (check metadata wiring).
- Ambiguous control names - ensure unique per form.
- SQL used in actions lacks required parameters (GGG/MMM, etc.).

Performance Guidance

- Cache small reference tables in RAM; avoid per-keystroke SQL calls.
- Keep grids lightweight (limit columns/rows) to speed redraw.
- Avoid excessive DOM toggling (Visible/Enabled) in a single event.

Security & Integrity

- Never embed user input directly into SQL - use parameterization patterns.
- Restrict EXEC targets to vetted stored procedures.
- Use Delete* only against RAM snapshots unless explicitly intended.

Testing Checklist

- [ ] Happy-path run with valid inputs
- [ ] Boundary values (min/max, empty, multi-response counts)
- [ ] Negative cases fire STOPIF/MSG as designed
- [ ] Focus & visibility shift as expected
- [ ] All assignments persist after Save/SaveNext
- [ ] Grid link actions perform and rollback cleanly

Quick Reference (Cheat Sheet)

- Condition fields: d.Field, #FP{Virtual}, constants '...'
- Multi-branch: separate parallel lists with $

- Skips/Deletes: pair targets and conditions 1:1
- Messages: SimpleMsgIs (OK) vs MsgIs (OK/Cancel)

Cross-References

- See: Assign, Stop, Skip, Visible, Enabled, ReadOnly, Exec, Delete, Message
- See controls: AutoCompleteTextBox, ComboBox, DataGridView, AlwaysVisibleLabel

Defaults

- ReadOnly=True, MultiSelect=False, AllowUserToAddRows/DeleteRows=False
- Order/resize columns & rows enabled; RowHeadersVisible=True
- EditMode=EditOnKeystroke; TabStop=False; Font=Verdana 8; Border=None

Examples

1) Simple read-only grid sourced from a view:

```
#{
GridSourceView=vw_PregledLica;
GridSourceWhere=ge1=d.ge1 and ge2=d.ge2;
GridOrderBy=Prezime, Ime;
GridColumns=Идентификатор=Id,Презиме=Prezime,Име=Ime,Статус=Status;
GridColumnsWidth=80,160,140,100;
GridRowsHeight=26;
GridTitle=Списак лица;
GridTitleFontBold;
GridHeaderRowFontSize=10;
NoTabStop;
}
```

2) Actionable grid with command columns:

```
#{
GridSource=EHISSpisakLica;
GridColumns=Рбр=Rbr,Презиме=Prezime,Име=Ime,Године=God;
GridLinkText=Уреди$Обриши$Отвори;
GridLinkAction=EditRowInGrid$Delete=EHISLica$Open=EHISLica15;
GridLinkWidth=80$80$80;
GridLinkStopIf=isnull(God,0)<15$$;       // block delete for <15, others allowed
GridLinkStopIfMsg=Не можете брисати за узраст < 15.$$;
EditRowInGridColumns=Prezime,Ime,God;
GridColumnsAutoResize;
GridTitle=Едит/Обриши/Отвори;
}
```

3) Data column as link, with Exec chain:

```
#{
GridSourceView=vw_Zakazi;
GridColumns=Датум=DatZak,Термин=Termin,Стање=Stanje;
GridColumnLink=DatZak$Termin;
```

GridColumnLinkAction=Open=Zakazivanja$Action#Exec;
GridTitle=Заказивања;
}
4) Add RAM-only columns and position them:
#{
GridSource=P2_Dom;
GridColumns=Рбр=RbrDom,Адреса=Adresa;
GridAddColumn=Лица=RAM.P2_DomCount,Валидација=RAM.ValInfo;
GridAddColumnOnPosition=2$3;
GridRowsHeight=24;
GridTitle=Домаћинства (са RAM колонама);
}
5) Conditional visibility and disabling with events:
#{
GridSource=Trace1;
GridColumns=Рбр=rbr,Догађај=ev,Опис=opis;
VisibleFalseIf=isnull(#FP{fpPrikaziTrag},0)=0;
EnabledFalseIf=isnull(#FP{fpZakljucano},0)=1;
OnSaveNextDeleteFrom=Trace1;
OnSaveNextDeleteIf=isnull(#FP{fpOdradiRefresh},0)=1;
GridTitle=Трагови (аутоматске радње на SaveNext);
}
6)Rooster - editable grid (ReadOnly=false)-
#{
DataGridView;
Rooster;
GridTitle=СПИСАК ПРОИЗВОДА/УСЛУГА;
GridLocation(X=20,Y=50);
GridSource=(select * from IKT_P16POM) a;
GridSourceWhere=god=d.god and mes=d.mes and MB=d.MB;
GridHeight=300;
GridWidth=400;
GridFontSize=7;
RowHeadersWidth=25;
GridHeaderRowFontSize=7;
ColumnHeadersHeight=70;
}
7) Command buttons column
#{
DataGridView;
GridTitle=Accounts;
GridLocation(X=200,Y=225);

GridSource=(select kpred, rbr, completename, username, admin,forDelete from v_UvedomAccounts_grd) a;
GridSourceWhere=kpred=d.kpred;
GRIDHeight=180;
GridWidth=400;
GridRowHeadersWidth=25;
GridColumnHeadersHeight=70;
GRIDFONTSIZE=7;
GRIDHeaderRowFONTSIZE=7;
GridColumns=Number Account=Rbr,Name=completeName,Account=UserName,Admin=Admin,Delete=forDelete;
GridColumnsWidth=60,130,130,130,130;
GridLinkText=✎;
GridLinkAction=Open=UvedomAccounts;
GridOrderBy=rbr;
GridColumnLink=forDelete;
GridColumnLinkAction=#exec usp_Brisi d.kpred,dgv.rbr;
}

Minimal Metadata Examples

#{
GridDataSource=(select top 100 * from Products order by title);
GridColumns=ID=id,Title=title,Price=price;
}

#{
GridSource=Codes;
GridOrderBy=name;
GridColumns=Code=code,Name=name;
GridLinkText=Open$Edit;
GridLinkAction=Open=Codes$EditRowInGrid;
}

AUTOCOMPLETETEXTBOX - FAST LIST-BACKED SELECTOR (TEXTBOX + LISTBOX)

Purpose

Offer fast, memory-backed typeahead from a file or SQL source. Supports cascading filters, assignments to other fields, and limit-to-list enforcement.

Syntax

#{
// From file
autoComplete=filename.txt;
autoCompleteStart={POS}; autoCompleteLength={LEN};
autoCompleteMinTypeLength={N}; autoCompleteLetter=LAT|CIR;

autoCompleteLimitToList=true|false;
autoCompleteAssignTo=Field1; autoCompleteAssignStart={POS}; autoCompleteAssignLength={LEN};

// From SQL
autoCompleteDataSource=(SELECT ...);
autoCompleteDisplayMember=colText; autoCompleteValueMember=colValue;
autoCompleteOrderBy=col1,col2; autoCompleteFilter=d.Field | #FP{Virtual};
autoCompleteMinTypeLength={N}; autoCompleteLimitToList=true;
}

Parameters

- File mode: Start/Length slice the file line for display and assignment.
- SQL mode: DisplayMember/ValueMember required; OrderBy optional.
- Filter: cascade by parent control or virtual fields.
- LimitToList: enforce selection; MinTypeLength controls popup activation.

Where to Define (Metadata Placement)

Define in _ISTRulesDataValidation.validationEvent for per-control rules, and/or in _ISTTablesColumns.columnAttributes for control-level properties. Form-level events (OnReturn*, OnSaveNext*, etc.) live in _ISTRulesFormEvents.

Execution Model

Evaluated by the .NET interpreter on control enter/leave and on form events. All conditions (...If) are boolean expressions against current RAM dataset (d.*) and virtual fields (#FP{...}). SQL fragments run against the active connection and respect the current ISTTimePoint (year/month) when applicable.

Data Types & Formatting

- Use cast/convert in expressions where numeric comparison is required.
- Multi-response values are comma-separated tokens (e.g., '1,3,5').
- String constants must be quoted with single quotes.

Interactions with Other Functions

- STOPIF/STOPMSG can short-circuit event flows before actions execute.
- ASSIGN/FOCUS/Visible*/Enabled*/ReadOnly* often used together post-validate.
- EXEC can be chained with grid actions or form-level events.

Best Practices

- Keep conditions explicit and side-effect free (no implicit conversions).
- Prefer field-safe guards (isnull(...)) to avoid null reference errors.
- Log critical event results with MsgIs to aid troubleshooting.

Common Errors & Diagnostics

- Condition evaluates but targets are missing from screen (check metadata wiring).
- Ambiguous control names - ensure unique per form.
- SQL used in actions lacks required parameters (GGG/MMM, etc.).

Performance Guidance

- Cache small reference tables in RAM; avoid per-keystroke SQL calls.
- Keep grids lightweight (limit columns/rows) to speed redraw.
- Avoid excessive DOM toggling (Visible/Enabled) in a single event.

Security & Integrity

- Never embed user input directly into SQL - use parameterization patterns.
- Restrict EXEC targets to vetted stored procedures.
- Use Delete* only against RAM snapshots unless explicitly intended.

Testing Checklist

- [ ] Happy-path run with valid inputs
- [ ] Boundary values (min/max, empty, multi-response counts)
- [ ] Negative cases fire STOPIF/MSG as designed
- [ ] Focus & visibility shift as expected
- [ ] All assignments persist after Save/SaveNext
- [ ] Grid link actions perform and rollback cleanly

Quick Reference (Cheat Sheet)

- Condition fields: d.Field, #FP{Virtual}, constants '...'
- Multi-branch: separate parallel lists with $
- Skips/Deletes: pair targets and conditions 1:1
- Messages: SimpleMsgIs (OK) vs MsgIs (OK/Cancel)

Cross-References

- See: Assign, Stop, Skip, Visible, Enabled, ReadOnly, Exec, Delete, Message
- See controls: AutoCompleteTextBox, ComboBox, DataGridView, AlwaysVisibleLabel

Examples

#{autocomplete=PROVINCES2.TXT;autocompletestart=6;autocompletelength=20;autocompletemintypelength=1;autocompleteassignto=A_1_13_4;autocompleteassignstart=2;autocompleteassignlength=2;autocompletelimittolist=True;autocompleteletter=LAT;}

#{autoCompleteDataSource=(select distinct kod,nname from vSPROPF_trade14); autoCompleteLimitToList=true; autoCompleteValueMember=kod; autoCompleteDisplayMember=nname; autocompleteMinTypeLength=1; autoCompleteLetter=CIR;}

#{autoCompleteDataSource=(select * from rpj.dbo.f_RPJ_Naselja({ggg},{mmm}));autoCompleteDisplayMember=nazivLOps;autoCompleteValueMember=mbops}

#{autoComplete=Drzave.txt;autoCompleteLimitToList=false;autoCompleteStart=5;autoCompleteLength=100;autoCompleteMinTypeLength=1;autoCompleteAssignTo=NAZIV_DRZAVE_SIFRA;autoCompleteAssignStart=0;autoCompleteAssignLength=2;}

Minimal Metadata Examples

#{autoCompleteDataSource=(select distinct nname, kod from vOKED3);
autoCompleteDisplayMember=nname; autoCompleteValueMember=kod;}

Relacija / WebRelacija / WebOnlyRelacija - Referential Lookup & Assignment

Purpose
Validate a parent record against a reference set (table or SQL) and optionally assign lookup columns to fields. If not found on entry, present a DataGridView for user selection. Web variants are used by the generator to build selection lists.

Syntax
#Relacija{
SomeTable | (SELECT ...);
p1,p2,...,pn: Assign1=Expr1, Assign2=Expr2[, ...] : [alwaysfromcodebook]
}
• Use #WebRelacija / #WebOnlyRelacija identically when targeting web generation.

Parameters
- SomeTable / (SELECT ...): reference dataset; supports {GGG},{MMM},{GGG±x},{MMM±x}, and D.Field in expressions.
- p1,p2,...,pn: key field(s) to match between current record and the reference dataset.
- Assignments: map reference columns or expressions to current form fields.
- alwaysfromcodebook: when present, assignment fields are always taken from reference even if DB row exists.

Execution Model (Details)
Batch validation: marks errors where NOT EXISTS in reference set for given keys. Data entry: checks in RAM; if no match, opens a grid bound to the reference set for the operator to choose a valid record. On match, executes assignments.

Examples
#Relacija{
(select * from RPJ.dbo.f_RPJ_Evs({GGG},{MMM}));
opsR=mbops: title3=titleMun, title2=titleMunCir
}
#Relacija{
(select * from products where od={GGG-5});
codeP2=sifpr: title=titleCN30
}
#Relacija{
(select * from products);
god=god, code3=code: title1=titleCN30
}
#Relacija{
(select * from viewView);
[mun]=codeMun, settlementTitle=titleLat: settl=codeSettl
}

Where to Define (Metadata Placement)
Define in _ISTRulesDataValidation.validationEvent for per-control rules, and/or in _ISTTablesColumns.columnAttributes for control-level properties. Form-level events (OnReturn*, OnSaveNext*, etc.) live in _ISTRulesFormEvents.

Execution Model
Evaluated by the .NET interpreter on control enter/leave and on form events. All conditions (...If) are boolean expressions against current RAM dataset (d.*) and virtual fields (#FP{...}). SQL fragments run against the active connection and respect the current ISTTimePoint (year/month) when applicable.

Data Types & Formatting
- Use cast/convert in expressions where numeric comparison is required.
- Multi-response values are comma-separated tokens (e.g., '1,3,5').
- String constants must be quoted with single quotes.

Interactions with Other Functions
- STOPIF/STOPMSG can short-circuit event flows before actions execute.
- ASSIGN/FOCUS/Visible*/Enabled*/ReadOnly* often used together post-validate.
- EXEC can be chained with grid actions or form-level events.

Best Practices
- Keep conditions explicit and side-effect free (no implicit conversions).
- Prefer field-safe guards (isnull(...)) to avoid null reference errors.
- Log critical event results with MsgIs to aid troubleshooting.

Common Errors & Diagnostics
- Condition evaluates but targets are missing from screen (check metadata wiring).
- Ambiguous control names - ensure unique per form.
- SQL used in actions lacks required parameters (GGG/MMM, etc.).

Performance Guidance
- Cache small reference tables in RAM; avoid per-keystroke SQL calls.
- Keep grids lightweight (limit columns/rows) to speed redraw.
- Avoid excessive DOM toggling (Visible/Enabled) in a single event.

Security & Integrity
- Never embed user input directly into SQL - use parameterization patterns.
- Restrict EXEC targets to vetted stored procedures.
- Use Delete* only against RAM snapshots unless explicitly intended.

Testing Checklist
- [ ] Happy-path run with valid inputs
- [ ] Boundary values (min/max, empty, multi-response counts)
- [ ] Negative cases fire STOPIF/MSG as designed
- [ ] Focus & visibility shift as expected

- [ ] All assignments persist after Save/SaveNext
- [ ] Grid link actions perform and rollback cleanly

Quick Reference (Cheat Sheet)
- Condition fields: d.Field, #FP{Virtual}, constants '...'
- Multi-branch: separate parallel lists with $
- Skips/Deletes: pair targets and conditions 1:1
- Messages: SimpleMsgIs (OK) vs MsgIs (OK/Cancel)

Cross-References
- See: Assign, Stop, Skip, Visible, Enabled, ReadOnly, Exec, Delete, Message
- See controls: AutoCompleteTextBox, ComboBox, DataGridView, AlwaysVisibleLabel

Minimal Metadata Examples
#Relacija{ Codes; code=code: title=title }

TextBox (Core) Single-Value Input with Validation

Purpose
Standard single-line or multi-line input. Supports Min/Max, MnLength/MxLength, Yes/No (allowed values), formatting, and common behaviors (ReadOnly, Enabled, Visible, Color).

Syntax
#{
Min={N}; Max={N};
MnLength={N}; MxLength={N};
Yes=numeric|alpha|... ; No=...;
Encrypt; Border*; fRightA|fCenterA; MultiLine;
ReadOnly; EnabledFalse; VisibleFalse;
}

Parameters
- Min/Max: numeric bounds; MnLength/MxLength: character bounds.
- Yes/No: whitelist/blacklist; Yes may be keyword (numeric, alpha) or explicit list.
- Styling & state: Encrypt, Border*, alignment flags, MultiLine.

Where to Define (Metadata Placement)
Define in _ISTRulesDataValidation.validationEvent for per-control rules, and/or in _ISTTablesColumns.columnAttributes for control-level properties. Form-level events (OnReturn*, OnSaveNext*, etc.) live in _ISTRulesFormEvents.

Execution Model
Evaluated by the .NET interpreter on control enter/leave and on form events. All conditions (...If) are boolean expressions against current RAM dataset (d.*) and virtual fields (#FP{...}). SQL fragments run against the active connection and respect the current ISTTimePoint (year/month) when applicable.

Data Types & Formatting

- Use cast/convert in expressions where numeric comparison is required.
- Multi-response values are comma-separated tokens (e.g., '1,3,5').
- String constants must be quoted with single quotes.

Interactions with Other Functions

- STOPIF/STOPMSG can short-circuit event flows before actions execute.
- ASSIGN/FOCUS/Visible*/Enabled*/ReadOnly* often used together post-validate.
- EXEC can be chained with grid actions or form-level events.

Best Practices

- Keep conditions explicit and side-effect free (no implicit conversions).
- Prefer field-safe guards (isnull(...)) to avoid null reference errors.
- Log critical event results with MsgIs to aid troubleshooting.

Common Errors & Diagnostics

- Condition evaluates but targets are missing from screen (check metadata wiring).
- Ambiguous control names - ensure unique per form.
- SQL used in actions lacks required parameters (GGG/MMM, etc.).

Performance Guidance

- Cache small reference tables in RAM; avoid per-keystroke SQL calls.
- Keep grids lightweight (limit columns/rows) to speed redraw.
- Avoid excessive DOM toggling (Visible/Enabled) in a single event.

Security & Integrity

- Never embed user input directly into SQL - use parameterization patterns.
- Restrict EXEC targets to vetted stored procedures.
- Use Delete* only against RAM snapshots unless explicitly intended.

Testing Checklist

- [ ] Happy-path run with valid inputs
- [ ] Boundary values (min/max, empty, multi-response counts)
- [ ] Negative cases fire STOPIF/MSG as designed
- [ ] Focus & visibility shift as expected
- [ ] All assignments persist after Save/SaveNext
- [ ] Grid link actions perform and rollback cleanly

Quick Reference (Cheat Sheet)

- Condition fields: d.Field, #FP{Virtual}, constants '...'
- Multi-branch: separate parallel lists with $
- Skips/Deletes: pair targets and conditions 1:1
- Messages: SimpleMsgIs (OK) vs MsgIs (OK/Cancel)

Cross-References

- See: Assign, Stop, Skip, Visible, Enabled, ReadOnly, Exec, Delete, Message
- See controls: AutoCompleteTextBox, ComboBox, DataGridView, AlwaysVisibleLabel

Minimal Metadata Examples
#{Min=1; Max=999; MnLength=1; MxLength=3; Yes=numeric;}

ComboBox / DropDown - Controlled Vocabulary Selector

Purpose
Classic dropdown bound to a table/view or arbitrary SQL, with optional filtering and limit-to-list. Use when the set of valid values is well-defined and relatively compact.

Syntax
#{
ComboDataSource={TABLE} | (SELECT ...);
ComboDisplayMember={TEXT_COL}; ComboValueMember={VALUE_COL};
ComboOrderBy={COLS}; ComboFilter=d.Field | #FP{Virtual};
ComboLimitToList=true|false; ComboRefreshOnEnter;
}

Parameters
- DataSource: table/view name, or SELECT statement.
- DisplayMember/ValueMember: text shown vs. value stored.
- OrderBy: comma-separated sort columns.
- Filter: cascade by current form values (d.*) or virtuals.

Where to Define (Metadata Placement)
Define in _ISTRulesDataValidation.validationEvent for per-control rules, and/or in _ISTTablesColumns.columnAttributes for control-level properties. Form-level events (OnReturn*, OnSaveNext*, etc.) live in _ISTRulesFormEvents.

Execution Model
Evaluated by the .NET interpreter on control enter/leave and on form events. All conditions (...If) are boolean expressions against current RAM dataset (d.*) and virtual fields (#FP{...}). SQL fragments run against the active connection and respect the current ISTTimePoint (year/month) when applicable.

Data Types & Formatting
- Use cast/convert in expressions where numeric comparison is required.
- Multi-response values are comma-separated tokens (e.g., '1,3,5').
- String constants must be quoted with single quotes.

Interactions with Other Functions
- STOPIF/STOPMSG can short-circuit event flows before actions execute.
- ASSIGN/FOCUS/Visible*/Enabled*/ReadOnly* often used together post-validate.
- EXEC can be chained with grid actions or form-level events.

Best Practices

- Keep conditions explicit and side-effect free (no implicit conversions).
- Prefer field-safe guards (isnull(...)) to avoid null reference errors.
- Log critical event results with MsgIs to aid troubleshooting.

Common Errors & Diagnostics

- Condition evaluates but targets are missing from screen (check metadata wiring).
- Ambiguous control names - ensure unique per form.
- SQL used in actions lacks required parameters (GGG/MMM, etc.).

Performance Guidance

- Cache small reference tables in RAM; avoid per-keystroke SQL calls.
- Keep grids lightweight (limit columns/rows) to speed redraw.
- Avoid excessive DOM toggling (Visible/Enabled) in a single event.

Security & Integrity

- Never embed user input directly into SQL - use parameterization patterns.
- Restrict EXEC targets to vetted stored procedures.
- Use Delete* only against RAM snapshots unless explicitly intended.

Testing Checklist

- [ ] Happy-path run with valid inputs
- [ ] Boundary values (min/max, empty, multi-response counts)
- [ ] Negative cases fire STOPIF/MSG as designed
- [ ] Focus & visibility shift as expected
- [ ] All assignments persist after Save/SaveNext
- [ ] Grid link actions perform and rollback cleanly

Quick Reference (Cheat Sheet)

- Condition fields: d.Field, #FP{Virtual}, constants '...'
- Multi-branch: separate parallel lists with $
- Skips/Deletes: pair targets and conditions 1:1
- Messages: SimpleMsgIs (OK) vs MsgIs (OK/Cancel)

Cross-References

- See: Assign, Stop, Skip, Visible, Enabled, ReadOnly, Exec, Delete, Message
- See controls: AutoCompleteTextBox, ComboBox, DataGridView, AlwaysVisibleLabel

Examples

```
#{
ComboDataSource=SIF_KD08;
ComboDisplayMember=delatnost;
ComboValueMember=kd08;
```

```
ComboOrderBy=delatnost;
}
#{
ComboDataSource=(select * from vOKED3 where left(kod,2)=left(d.okd_3,2));
ComboDisplayMember=nname; ComboValueMember=kod; ComboLimitToList=true;
}
```

Minimal Metadata Examples

```
#{
ComboDataSource=(select distinct kod,nname from vSPROPF_trade14);
ComboDisplayMember=nname; ComboValueMember=kod; ComboLimitToList=true;
}
```

ALWAYSVISIBLELABEL - CONTEXTUAL HELP LABEL

Purpose

A helper label (System.Windows.Forms.Label) that appears when a control receives focus or the user presses F1, and hides on leave. Commonly used to present full code lists for MultiResponse fields or richer instructions.

Syntax

```
#{
AssignTo=AlwaysVisibleLabel;
AssignWhat='<formatted help text or code list>';
VisibleTrueIf={COND}; VisibleFalseIf={COND}; // optional
}
```

Parameters

- AssignTo: must be AlwaysVisibleLabel to target the helper label.
- AssignWhat: rich-text payload (supports <bold>, <fontsize>,
).
- Optional styling via Font*, foreColor/backColor on the label control.

Where to Define (Metadata Placement)

Define in _ISTRulesDataValidation.validationEvent for per-control rules, and/or in _ISTTablesColumns.columnAttributes for control-level properties. Form-level events (OnReturn*, OnSaveNext*, etc.) live in _ISTRulesFormEvents.

Execution Model

Evaluated by the .NET interpreter on control enter/leave and on form events. All conditions (...If) are boolean expressions against current RAM dataset (d.*) and virtual fields (#FP{...}). SQL fragments run against the active connection and respect the current ISTTimePoint (year/month) when applicable.

Data Types & Formatting

- Use cast/convert in expressions where numeric comparison is required.
- Multi-response values are comma-separated tokens (e.g., '1,3,5').
- String constants must be quoted with single quotes.

Interactions with Other Functions

- STOPIF/STOPMSG can short-circuit event flows before actions execute.
- ASSIGN/FOCUS/Visible*/Enabled*/ReadOnly* often used together post-validate.
- EXEC can be chained with grid actions or form-level events.

Best Practices

- Keep conditions explicit and side-effect free (no implicit conversions).
- Prefer field-safe guards (isnull(...)) to avoid null reference errors.
- Log critical event results with MsgIs to aid troubleshooting.

Common Errors & Diagnostics

- Condition evaluates but targets are missing from screen (check metadata wiring).
- Ambiguous control names - ensure unique per form.
- SQL used in actions lacks required parameters (GGG/MMM, etc.).

Performance Guidance

- Cache small reference tables in RAM; avoid per-keystroke SQL calls.
- Keep grids lightweight (limit columns/rows) to speed redraw.
- Avoid excessive DOM toggling (Visible/Enabled) in a single event.

Security & Integrity

- Never embed user input directly into SQL - use parameterization patterns.
- Restrict EXEC targets to vetted stored procedures.
- Use Delete* only against RAM snapshots unless explicitly intended.

Testing Checklist

- [ ] Happy-path run with valid inputs
- [ ] Boundary values (min/max, empty, multi-response counts)
- [ ] Negative cases fire STOPIF/MSG as designed
- [ ] Focus & visibility shift as expected
- [ ] All assignments persist after Save/SaveNext
- [ ] Grid link actions perform and rollback cleanly

Quick Reference (Cheat Sheet)

- Condition fields: d.Field, #FP{Virtual}, constants '...'
- Multi-branch: separate parallel lists with $
- Skips/Deletes: pair targets and conditions 1:1
- Messages: SimpleMsgIs (OK) vs MsgIs (OK/Cancel)

Cross-References

- See: Assign, Stop, Skip, Visible, Enabled, ReadOnly, Exec, Delete, Message
- See controls: AutoCompleteTextBox, ComboBox, DataGridView, AlwaysVisibleLabel

Examples

#{
AssignTo=AlwaysVisibleLabel;
AssignWhat='<fontsize10><bold>12. Држављанство</bold>
уписати назив државе...';
}
#{
AssignTo=AlwaysVisibleLabel;
AssignWhat='Enter 1–3 options separated by comma. Press F1 for help.';
}

Minimal Metadata Examples

#{
AssignTo=AlwaysVisibleLabel; AssignWhat='Codes: 1 A, 2 B, 3 C';
}

VALIDATION PATTERNS CATALOG

Purpose

Provide a reusable recipe book for composing soft validations (range, length, allowed values) and hard validations (StopIf/StopMsg), including specialized MultiResponse constraints. These patterns standardize how constraints are layered, ordered, and messaged to end users.

Where to Define (Metadata Placement)

- _ISTTablesColumns.ValidationEvent - primary location for control-level validation rules.
- _ISTRulesDataValidation - table-level or cross-field validations that need batch checks.
- _ISTMessages - centralize reusable messages referenced by StopMsg/StopMsgOnPnl.

Syntax

#{
Yes=Numeric; Min=0; Max=999;
}
#{
StopIf=amount>limit; StopMsg='Amount exceeds limit';
}

MultiResponse constraints with exact count:

#{
Yes=MultiResponseNumeric; Min=1; Max=12;
MnResponseCount=5; MxResponseCount=5;
}

Parameters

- Soft checks: Min, Max, MNLength, MXLength, Yes, No, AWLValue.
- Hard checks: StopIf, StopMsg, StopMsgOnPnl.
- MultiResponse: Yes=MultiResponseAlpha|MultiResponseNumeric, MnResponseCount, MxResponseCount.

Execution Model

- Soft checks run first to constrain domain and shape input.
- Hard checks (StopIf/StopMsg) run after soft checks to block invalid combinations.
- For MultiResponse fields, response-count validation is evaluated before logical combination rules.

Data Types & Formatting

- Yes=Numeric/Text/MultiResponseNumeric/MultiResponseAlpha to set the expected type.
- No= disallow list for MultiResponse; values separated by commas.
- Use MNLength/MXLength for text length; Min/Max for numeric ranges.

Interactions with Other Functions

- Skip: branch to follow-up questions after valid combos.
- AlwaysVisibleLabel: display code lists and instructions alongside MultiResponse controls.
- Message: use MsgIs/OptionalMsgIs for non-blocking guidance; StopMsg for hard stops.

Examples

1) Soft then hard:

```
#{
Yes=Numeric; Min=0; Max=999;
}
#{
StopIf=amount>limit; StopMsg='Amount exceeds limit';
}
```

2) MultiResponse with exact count:

```
#{
Yes=MultiResponseNumeric; Min=1; Max=12;
MnResponseCount=5; MxResponseCount=5;
}
```

3) Disallow mixed categories:

```
#{
StopIf=isnull(ans,'') like '%3%' and (isnull(ans,'') like '%1%' or isnull(ans,'') like '%2%');
StopMsg='Category 3 cannot be mixed with 1 or 2';
}
```

Best Practices

- Layer: soft checks → count checks (for MultiResponse) → hard logic checks.
- Make messages action-oriented and specific; avoid generic 'Invalid input'.
- For MultiResponse, always set both MnResponseCount and MxResponseCount when exact cardinality is required.

Common Errors & Diagnostics

- Hard rule triggers before soft constraints are satisfied (reorder blocks).
- Incorrect wildcard usage in LIKE patterns for MultiResponse matching.
- Missing entries in _ISTMessages for StopMsgOnPnl keys.

Performance Guidance

- Consolidate simple soft checks into one block per control.
- Prefer deterministic count checks over complex LIKE patterns where possible.

Security & Integrity

- Keep message text non-sensitive; don't echo PII in StopMsg.
- Avoid dynamic SQL in validation expressions; use whitelisted functions only.

Testing Checklist (Copy/Paste)

[ ] Min/Max and MN/MXLength enforced
[ ] MultiResponse exact/at-least/at-most cases
[ ] Hard StopIf fires with correct StopMsg
[ ] No/Yes allowlists behave as expected
[ ] Labels show complete instruction set

Quick Reference (Cheat Sheet)

Soft: Yes/No/AWLValue, Min/Max, MN/MXLength
Hard: StopIf + StopMsg, StopMsgOnPnl
MultiResponse: Yes=MultiResponse*, MnResponseCount/MxResponseCount

Cross-References

- Message, Skip, AlwaysVisibleLabel, MultiResponseTextBox

Minimal Metadata Examples

```
#{
Yes=Numeric; Min=0; Max=100;
}
#{
StopIf=value>threshold; StopMsg='Over threshold';
}
#{
Yes=MultiResponseNumeric; MnResponseCount=3; MxResponseCount=3;
}
```

LOCALIZATION AND SCRIPTS (LAT/CIR)

Purpose

Enable bilingual forms and script-appropriate codebooks/inputs (Latin/Cyrillic), with per-control font overrides and optional bilingual titles.

Where to Define (Metadata Placement)

- _ISTTablesColumns.ColumnAttributes - control-level script and font settings (e.g., autoCompleteLetter).
- _ISTTablesColumns.Title / Title2 / Title3 - multi-language label variants when supported.

Syntax

#{
autoCompleteLetter=LAT|CIR; autoCompleteLimitToList=true|false;
autoCompleteMinTypeLength=Number; Font*=...
Title2/Title3=...
}

Parameters

- autoCompleteLetter=LAT|CIR - constrain lookup/input script for AutoComplete/Combo.
- autoCompleteMinTypeLength - characters required before lookup.
- Font*, label font sizes and styles per control.
- Title2/Title3 - alternate label texts (if control supports).

Execution Model

- Script constraint applies at input-time and to lookup rendering when using AutoCompleteTextBox/Combo.
- Font overrides render immediately on control paint; no data impact.

Data Types & Formatting

- Script constraint affects displayed glyph set; underlying values remain unchanged.
- Use matching codebooks for LAT/CIR when LimitToList=true.

Interactions with Other Functions

- AutoCompleteTextBox/ComboBox - primary consumers of script settings.
- AlwaysVisibleLabel - display bilingual guidance next to inputs.
- Message - notify users when script doesn’t match expectation.

Examples

1) Force Cyrillic lookup:

#{
autoCompleteLetter=CIR; autoCompleteLimitToList=true;
}

2) Latin search with minimum length:

#{
autoCompleteLetter=LAT; autoCompleteMinTypeLength=1;
}

Best Practices

- Keep codebooks synchronized for both scripts to prevent inconsistent results.
- Use LimitToList=true to avoid mixed-script free-text entries when standardization is critical.

Common Errors & Diagnostics

- Misaligned codebooks (LAT vs. CIR) causing missing hits.
- MinTypeLength set too high causing user confusion.

Performance Guidance

- Leverage indexed views for lookups; avoid heavy joins in AutoCompleteDataSource.
- Keep MinTypeLength at 1–2 to reduce query load.

Security & Integrity

- Localize labels without embedding sensitive content in metadata.
- Validate server-side that values map to expected codebook entries.

Testing Checklist (Copy/Paste)

[ ] Script toggle LAT/CIR behaves correctly
[ ] Correct codebook returned per script
[ ] LimitToList prevents off-list entries
[ ] Fonts render legibly for both scripts

Quick Reference (Cheat Sheet)

Script: autoCompleteLetter=LAT|CIR
Limit: autoCompleteLimitToList=true|false
UX: autoCompleteMinTypeLength=1..n, Font*, Title2/Title3

Cross-References

- AutoCompleteTextBox, ComboBox_DropDown, AlwaysVisibleLabel

Minimal Metadata Examples

#{autoCompleteLetter=CIR; autoCompleteLimitToList=true;}
#{autoCompleteLetter=LAT; autoCompleteMinTypeLength=1;}

FILEPICKERS & ATTACHMENTS

Purpose

Provide standardized patterns for letting users select or attach files to a record, validate what they picked, and optionally post-process (store/move/scan) the file. The feature is project-dependent - include only when your application actually supports file attachments.

Where to Define (Metadata Placement)

Place validation and behavior rules primarily in the validation event field for the file path control (e.g., _ISTTablesColumns.columnAttributes). Messaging text can live in ISTMessages for reuse. If server-side post-processing is required, the stored procedure and parameters are referenced in the form-level or control-level rules.

Syntax

#{
Yes=Alpha|Numeric|FilePath;
MNLength=integer; MXLength=integer;
StopIf=<blacklist or constraint condition>;
StopMsg='Message to user';
OnSaveNextExec=exec usp_store_file d.rbr, d.filePath;
}

Parameters

- Yes=FilePath - control accepts a file path (or control-specific flags).
- MNLength/MXLength - Minimum/maximum length of the entered path.
- StopIf / StopMsg - Validation guardrail (extension, size, path rules).
- OnSaveNextExec - Post-processing (store/move/scan, write metadata).

Execution Model

Validation runs on control leave and again on Save/SaveNext. If StopIf is satisfied, Save is blocked and StopMsg is shown. On SaveNext, optional OnSaveNextExec runs to persist/transform the file reference.

Data Types & Formatting

- Expected input: absolute or project-defined paths; normalize slashes if needed.
- Extensions should be checked case-insensitively.
- If stored, save a URI or normalized path, not raw user input.

Interactions with Other Functions

- Messaging Panels: Provide user guidance before selection and clear error text upon failure.
- Security Rules: Enforce PII/PHI constraints and blacklist risky extensions.
- OnSaveNext*: Chain post-processing with transaction-scoped stored procedures.

Examples

Basic whitelist by extension

```
#{
Yes=Alpha; MNLength=3;
StopIf=right(filePath,4) not in ('.pdf','.jpg');
StopMsg='Allowed: PDF/JPG';
}
```

Post-process on SaveNext

```
#{
OnSaveNextExec=exec usp_store_file d.rbr, d.filePath;
}
```

Best Practices

- Whitelist extensions; avoid lax wildcards.
- Keep messages actionable (what to fix, allowed options).
- Never store full local paths if privacy is a concern; prefer server location aliases.

Common Errors & Diagnostics

- StopIf too permissive: risky files slip through.
- StopIf too strict: valid files blocked; review extension list casing and dots.
- Stored procedure errors: log name and parameters; ensure permissions.

Performance Guidance

Perform file validation client-side quickly (length, extension). Defer heavy scanning or storage to server-side asynchronous procedures triggered by OnSaveNextExec where possible.

Security & Integrity

- Sanitize file names; block executable content.
- Scan uploads server-side; validate MIME types if supported.
- Avoid logging sensitive file paths; store hashed references when possible.

Testing Checklist (Copy/Paste)

[ ] Accepts only whitelisted extensions
[ ] Blocks blacklisted/oversized files with clear message
[ ] Normalizes stored path/URI
[ ] OnSaveNextExec fires with correct params
[ ] Graceful failure if SP not available
[ ] Messages localized where required

Quick Reference (Cheat Sheet)

Basic validation
#{ Yes=Alpha; MNLength=3; StopIf=right(filePath,4) not in ('.pdf','.jpg'); StopMsg='Allowed: PDF/JPG'; }
Post-process
#{ OnSaveNextExec=exec usp_store_file d.rbr, d.filePath; }

Cross-References

- Messaging Panels - for pre/post guidance.
- Security & Privacy - extension/MIME governance.
- OnSaveNextForm - post-save hooks.

Minimal Metadata Examples

#{
Yes=Alpha;
StopIf=right(filePath,4) not in ('.pdf','.png','.jpg');
StopMsg='Upload PDF/PNG/JPG only';
}

THEMING & COLORING

Purpose

Apply consistent, accessible color semantics across controls and containers to guide user attention (required, error, warning, info) and support visual hierarchy.

Where to Define (Metadata Placement)

Define ColorTo/ColorWhat/ColorIf in the control/section rule blocks (_ISTTablesColumns.columnAttributes or form-level rules). Use shared constants for named colors when available.

Syntax

#{
ColorTo={COLOR_NAME};
ColorWhat={ControlOrContainer|Group|AllReadOnly};
ColorIf={BooleanCondition};
}

Parameters

- ColorTo - Named color (e.g., LightYellow, MistyRose, Gainsboro).
- ColorWhat - Target: single control, container/section, or semantic group (e.g., AllReadOnly).
- ColorIf - Condition under which the color is applied.

Execution Model

Evaluated on form load and whenever the referenced fields in ColorIf change. Rendering should be idempotent and revert when condition is false.

Data Types & Formatting

- Color names map to System.Drawing/KnownColor (or project palette).
- Prefer semantic groups over hard-coded control lists when possible.

Interactions with Other Functions

- ReadOnly/Enabled - pair de-emphasis with non-editable state.
- Messaging Panels - visually reinforce validation results.
- Visible - avoid coloring controls that are hidden by logic.

Examples

Required highlight on empty field

```
#{
ColorTo=LightYellow; ColorWhat=code; ColorIf=isnull(code,'')='';
}
```

Error state for a whole section

```
#{
ColorTo=MistyRose; ColorWhat=SectionA; ColorIf=cast(isnull(hasErrors,0) as int)=1;
}
```

De-emphasize read-only blocks

```
#{
ColorTo=Gainsboro; ColorWhat=AllReadOnly; ColorIf=isnull(readonly,0)=1;
}
```

Best Practices

- Follow WCAG contrast ratios; do not convey meaning by color alone.
- Use a small, documented palette to reduce cognitive load.
- Reset styles when conditions no longer apply.

Common Errors & Diagnostics

- Conflicting color rules (last write wins) - consolidate scopes.
- Over-coloring reduces clarity - reserve strong colors for errors.
- Missing reversion logic - ensure conditions are mutually exclusive or ordered.

Performance Guidance

Batch UI updates when multiple coloring rules depend on the same fields to avoid flicker.

Security & Integrity

No direct security impact; ensure color states never mask required fields or errors.

Testing Checklist (Copy/Paste)

[ ] Required fields get highlight when empty
[ ] Error sections turn to error color and revert when fixed
[ ] Read-only blocks are de-emphasized consistently
[ ] High contrast retained in all themes
[ ] No color applied when controls are hidden

Quick Reference (Cheat Sheet)

```
# required on empty
#{ ColorTo=LightYellow; ColorWhat=code; ColorIf=isnull(code,'')=''; }

# error container
#{ ColorTo=MistyRose; ColorWhat=SectionA; ColorIf=cast(isnull(hasErrors,0) as int)=1; }

# readonly de-emphasis
#{ ColorTo=Gainsboro; ColorWhat=AllReadOnly; ColorIf=isnull(readonly,0)=1; }
```

Cross-References

- ReadOnly - lock state aligned with de-emphasis.
- Enable - dynamic enable/disable patterns.
- Messaging Panels - coordinate color with messages.

Minimal Metadata Examples

```
#{
ColorTo=LightYellow; ColorWhat=customerName; ColorIf=isnull(customerName,'')='';
}
```

11. Developer Troubleshooting and Diagnostics

This section provides practical guidance to developers on how to identify, isolate, and resolve common issues encountered during the development, testing, and deployment of IST-based applications. It includes information on runtime behavior, error logging, metadata validation, and debugging techniques for desktop, web, and Android environments.

11.1 Common Metadata Errors

Symptom	Possible Cause	Solution
Form does not open	Missing or misaligned ValidFrom/ValidTo in metadata tables	Ensure metadata is valid for selected time point.
Field not visible on form	Field missing from _ISTTableColumns or bad column name	Verify column exists and is valid.
Validation not triggered	Logic placed in wrong field or missing 'condition'	Use correct metadata field and validation trigger.
ComboBox is empty	Missing or incorrect relational table/column	Check consult configuration.
Virtual field shows no value	Expression missing or invalid	Test expression in SSMS before metadata insertion.
Messages not localized	Missing _ISTMessages or _ISTLabels	Add entries for target language.

11.2 Runtime Metadata Validation Tips

- Check ValidFrom and ValidTo in all relevant tables.
- Ensure all tables and columns exist in SQL Server.
- Validate expressions in SSMS before placing in metadata.

Example SQL:
SELECT * FROM _ISTTables WHERE appCode = 'XYZApp' AND ValidFrom <= GETDATE() AND (ValidTo IS NULL OR ValidTo >= GETDATE())

11.3 Debugging Form Behavior in IST

- Use Event Viewer or SQL Profiler for application errors.
- Add label-type VirtualFields to display expression outputs.
- Use ExecMsgIs or temporary messages for debugging logic.

11.4 Useful Logs and Tables for Troubleshooting

Log/Table	Use	Location
_ISTLogDelete	Tracks deleted records	DEPO
_ISTVariablesChange	Tracks data edits per variable	DEPO
_ISTLogBatchLC	Logs batch validations	DEPO
_ISTLogProcessUsage	Tracks report executions	DEPO

_ISTSavedAdvancedSearch	Stores query SQL from Advanced Search	DEPO

11.5 SQL Profiler Tips (Advanced)

- Track loading of metadata.
- Identify long-running expressions.
- Filter by Application Name = ISTInterpreter.

11.6 Best Practices for Debugging Virtual Fields

- Break complex expressions into smaller parts.
- Use hardcoded debug values for testing logic.
- Use :PK and :CalculateOnExit where applicable.

11.7 Tips for Testing in Controlled Environments

- Clone metadata for test apps.
- Use time point like 2025/01 for sandboxing.
- Deploy locally for debugging with logs.
- Maintain dedicated DEV metadata database.

11.8 Intermittent Errors: Causes and Resolutions

Issue	Likely Cause	Fix
Virtual fields sometimes blank	Improper dependency setup	Use :CalculateOnExit=FieldX
ComboBox values missing	Metadata not valid for time point	Adjust ValidFrom/ValidTo
Fields not updating	Expression error or wrong event	Place logic in correct metadata field
Batch validation slow	Inefficient WHERE clause	Optimize SQL or use indexes

Appendix 1 – Full Example of Metadata Setup in IST

This appendix provides a complete example of a minimal yet functional IST application, covering all required metadata tables and logical elements needed to create a simple statistical survey. The example includes a parent-child table structure, virtual fields, validation rules, and a report definition.

1. Scenario Overview

We are building an application for collecting data on households and their members. The application consists of two tables:
- `Households` (master table)
- `Members` (child table)

2. Required Metadata Tables and Example Entries

2.1 _IST

appCode: DEMOAPP
appTitle: Demo – Household Survey
databaseAlias: DEMODATA
additionalParameters: IST=DEMOAPP
ValidFrom: 2025-01-01
ValidTo: NULL

2.2 _ISTDatabaseConnStrings

databaseAlias: DEMODATA
ODBC: Data Source=localhost;Initial Catalog=DemoDB;Integrated Security=SSPI;
ValidFrom: 2025-01-01
ValidTo: NULL

2.3 _ISTTables

appCode: DEMOAPP
tableName: dbo.Households
typeOfTableParentChild: G
tableDescription: Households
ValidFrom: 2025-01-01

appCode: DEMOAPP
tableName: dbo.Members
typeOfTableParentChild: D
parentTableName: dbo.Households
dataEntryScreenOrder: 1
tableDescription: Members

ValidFrom: 2025-01-01

2.4 _ISTTableColumns

appCode: DEMOAPP
tableName: dbo.Households
columnName: HHID
primaryKey: P
columnType: int
columnLength: 4
label: Household ID
orderNumber: 1
ValidFrom: 2025-01-01

appCode: DEMOAPP
tableName: dbo.Members
columnName: HHID
primaryKey: P
columnType: int
columnLength: 4
label: Household ID (FK)
orderNumber: 1
ValidFrom: 2025-01-01

columnName: MemberID
primaryKey: P
columnType: int
columnLength: 4
label: Member ID
orderNumber: 2
ValidFrom: 2025-01-01

columnName: Age
columnType: int
columnLength: 3
label: Age
orderNumber: 3
validatingEvent: #if(d.Age<0 or d.Age>120) {error('Invalid age')}
ValidFrom: 2025-01-01

columnName: IsAdult
columnType: bit
columnLength: 1
label: Is Adult?

columnAttributes: #FP{IsAdult}=iif(d.Age>=18,1,0)
orderNumber: 4
ValidFrom: 2025-01-01

2.5 _ISTRulesDataValidation

appCode: DEMOAPP
tableName: dbo.Members
errNumber: e1
Error: Age < 0 OR Age > 120
condition: ExecuteOnlyOnDataEntryForm
errTitle: Age must be between 0 and 120
errWeight: T
ValidFrom: 2025-01-01

2.6 _ISTReportsProcedures

appCode: DEMOAPP
SeqNumber: 1
Title: Members Summary
query: select HHID, count(*) as TotalMembers, avg(Age) as AvgAge from dbo.Members group by HHID
ValidFrom: 2025-01-01

3. Expected Behavior

The application allows:
- Entry of households and their members.
- Automatic validation of member age.
- Computation of a virtual field `IsAdult` based on age.
- Execution of a summary report grouped by household.

Appendix 2 – Quick functions summary

Category	Function / Directive	Purpose
Validations	Min, Max	Enforce inclusive numeric bounds
Validations	Yes=...; No=...	Type/domain: numeric, integer, MultiResponseNumeric, etc.
Validations	MnLength, MxLength	Enforce min/max string length
Validations	StopIf, StopMsg	Blocking condition and message
Validations	RequiredIf	Conditionally require a value
Validations	Warning, SimpleMsgIf, SimpleMsgIs	Non-blocking alert conditions
Skips	SkipTo, SkipIf	Conditional navigation to targets
Skips	SkippedEnabledFalse	Disable controls that were bypassed
Skips	SkippedSetEmpty, SkippedSetEmptyExcept, SkippedSetEmptyExceptIf	Clear skipped values with exceptions
Skips	SkippedDontSave	Do not persist values from skipped fields to the database
Skips	SkipAction	Per-branch actions on skipped fields (SetEmpty, EnabledFalse, VisibleFalse, DontSave)
Controls	VisibleTrue/False, VisibleTrueIf/FalseIf	Show/hide controls conditionally
Controls	EnableFalse/True, EnableFalseIf/TrueIf	Disable/enable controls conditionally
Controls	ReadOnly, ReadOnlyIf	Make controls read-only conditionally
Assignments	AssignTo, AssignWhat, AssignIf	Auto-fill/normalize values
Mutations	deleteFrom, deleteWhere, deleteIf	Delete/cascade rows when conditions change
Virtual Fields	#FP{Field}	Runtime-calculated fields
Batch Rules	ISTRulesDataValidation	Table-level validations with SQL conditions
Tokens/SQL	#MOD11{}, isnull, cast, len, exists, in, like, isDate	Checksum and SQL utility functions

1. Introduction & Best Practices

- Metadata-first: keep all logic in metadata; avoid hardcoding in UI code.
- Guard for nulls: wrap references with isnull() to avoid runtime surprises.
- Be explicit: prefer explicit StopIf for complex rules even when Min/Max or Mn/MxLength exist.
- Single source of truth: avoid contradictory directives on the same control.
- Idempotency: ensure assignments do not oscillate values.
- Test pathways: cover happy path, edge cases, and error handling.

2. Validations

2.1 Range & Type

Purpose: Ensure values meet numeric ranges and expected types.

Minimal:

```
#{
 Min=0; Max=120; Yes=numeric;
 StopMsg='Age must be between 0 and 120';
}
```

Explicit StopIf (alternative or combined):

```
#{
 StopIf=isnull(Age,0)<0 or isnull(Age,0)>120;
 StopMsg='Age must be between 0 and 120';
}
```

Type enforcement:

```
#{
 Yes=integer;
 StopMsg='Only whole numbers are allowed';
}
```

2.2 Length & Patterns

Fixed length numeric ID (13 digits):

```
#{
 Yes=numeric; MnLength=13; MxLength=13;
 StopIf=len(isnull(JMBG,''))<>13;
 StopMsg='JMBG must be exactly 13 digits';
}
```

Date format (dd.MM.yyyy):

```
#{
 Yes=istDate;
 StopIf=isnull(DateField,'')<>'' and isDate(DateField)=0;
 StopMsg='Use format dd.MM.yyyy';
}
```

2.3 Conditional Required

Required when status is Active:

```
#{
 RequiredIf=Status='Active';
 StopIf=isnull(ActiveDate,'')='';
 StopMsg='Active Date is required when Status is Active';
}
```

At least one contact provided:

```
#{
 StopIf=isnull(Phone,'')='' and isnull(Email,'')='';
 StopMsg='Provide either Phone or Email';
}
```

2.4 Multi-Response

1–3 selections from 1..10:

```
#{
 Yes=MultiResponseNumeric; Min=1; Max=10;
 MnResponseCount=1; MxResponseCount=3;
 StopIf=ResponseCount<1 or ResponseCount>3;
 StopMsg='Select between 1 and 3 options';
}
```

Disallow conflicting choices:

```
#{
 StopIf=isnull(Answers,'') like '%3%' and (isnull(Answers,'') like '%1%' or isnull(Answers,'') like '%2%');
 StopMsg='Option 3 cannot be combined with 1 or 2';
}
```

2.5 Cross-Field & Cross-Table

End > Start:

```
#{
 StopIf=EndDate<StartDate;
 StopMsg='End date must be after start date';
}
```

Value must exist in reference table:

```
#{
 StopIf=not exists(select 1 from RefTable where RefCode=InputCode);
 StopMsg='Code not found in reference list';
}
```

2.6 Warnings (Soft Validations)

Non-blocking outlier alert:

```
#{
 Warning;
 SimpleMsgIf=Value<10 or Value>90;
 SimpleMsgIs='Value is outside the typical range - please confirm.';
}
```

3. Skips

3.1 Basic Skip Model

```
#{
```

```
 SkipTo=NextQuestion;
 SkipIf=HasChildren=0;
 SkippedEnabledFalse;
 SkippedSetEmpty;
}
```

3.2 Multi-Branch & Exceptions

```
#{
 SkipTo=Q41$Q37_7;
 SkipIf=6 not in Pomocno_G4$6 in Pomocno_G4;
 SkippedEnabledFalse;
 SkippedSetEmpty;
}
```

With exceptions (preserve certain fields):

```
#{
 SkipTo=BrojDom$UBrLicaStan$ButtonNext;
 SkipIf=cast(isnull(KoZiviuStanu,0) as int)=1$cast(isnull(KoZiviuStanu,0) as
int)=2$cast(isnull(KoZiviuStanu,0) as int) in (3,4,5,6,7);
 SkippedEnabledFalse;
 SkippedSetEmptyExcept=BrojDom$BrojDom,UBrLicaStan;
 SkippedSetEmptyExceptIf=cast(isnull(KoZiviuStanu,0) as int)=2$cast(isnull(KoZiviuStanu,0) as int) in
(3,4,5,6,7);
}
```

3.3 Skip Side-Effects (Disable/Clear)

- SkippedEnabledFalse: disable skipped controls.
- SkippedSetEmpty: clear values in skipped controls.
- Exception forms prevent clearing/disable on protected fields.

3.4 SkipAction - Common Values and Combinations

Purpose: Define what happens to skipped fields when a skip condition is triggered. Ensures data integrity and UI consistency by applying one or more actions.

Individual actions:

- SetEmpty - Clears the value of the skipped field(s).
- EnabledFalse - Disables skipped fields (non-editable).
- VisibleFalse - Hides skipped fields from the UI.
- DontSave - Prevents saving values from skipped fields to the database.

Combining actions (syntax):

- Use $ to separate actions per branch, matching the order of SkipTo/SkipIf branches.
- Within a branch, separate multiple actions with commas (,).

One-branch example (multiple actions in the same branch):

```
#{
 SkipTo=NextSection;
 SkipIf=HasChildren=0;
 SkipAction=SetEmpty,EnabledFalse,VisibleFalse,DontSave;
}
```

Two-branch example (actions aligned to branches):

```
#{
 SkipTo=A_1_13_4$buttonnext;
 SkipIf=A_1_4_4>2$A_1_4_4 in (1,2);
 SkipAction=SetEmpty,EnabledFalse$DontSave,VisibleFalse;
}
```

Interpretation: first branch clears and disables; second branch does not save and hides fields.

Per-branch single actions (shorthand):

```
#{
 SkipTo=Q1$Q2$Q3;
 SkipIf=Cond1$Cond2$Cond3;
 SkipAction=SetEmpty$EnabledFalse$VisibleFalse;
}
```

- Align the number of $ segments in SkipAction with SkipTo/SkipIf branches.
- When in doubt, use explicit comma-separated actions within each branch for readability.
- Combine DontSave with SetEmpty when you need strict data integrity.
- Prefer VisibleFalse for a cleaner UI if fields are irrelevant after skip.

4. Controls

4.1 Visibility

```
#{
 VisibleTrue=ReasonText;
 VisibleTrueIf=cast(isnull(HasReason,0) as int)=1;
}
```

4.2 Enable/Disable

```
#{
 EnableFalse=AllInputs;
 EnableFalseIf=cast(isnull(IsSubmitted,0) as int)=1;
}
```

4.3 Read-Only

```
#{
 ReadOnly=MBR,OPS;
 ReadOnlyIf=cast(isnull(IsImported,0) as int)=1;
}
```

5. Assignments & Mutations

5.1 Assignments

Initialize status when empty:

```
#{
 AssignTo=Status;
 AssignWhat='NEW';
 AssignIf=isnull(Status,'')='';
}
```

Normalize gender code from label:

```
#{
 AssignTo=GenderCode;
 AssignWhat=case when Gender='Male' then 1 when Gender='Female' then 2 else null end;
 AssignIf=1=1;
}
```

5.2 Deletions & Cascades

```
#{
 deleteFrom=Trace1,Trace2,Lica;
 deleteWhere=rbr>cast(#FP{fprbr} as integer);
 deleteIf=#FP{fprbrTRACE1}>#FP{fprbr};
}
```

Best practice: always use a narrow deleteWhere predicate and test on a copy.

6. Virtual Fields

6.1 #FP{} Expressions

IsAdult virtual:

```
#FP{IsAdult}=case when isnull(d.Age,0)>=18 then 1 else 0 end
```

6.2 SQL-Derived Values

Household member full name:

```
#FP{MemberFullName}=ltrim(rtrim(isnull(d.FirstName,'')+' '+isnull(d.LastName,'')))
```

Usage in validations/skips:

```
#{
 SkipTo=EmploymentSection;
 SkipIf=#FP{IsAdult}=1;
}
```

7. Batch Rules (Table-Level)

Rule columns: appCode, tableName, errNumber, Error (SQL condition), errTitle, errWeight, ValidFrom.

Example (SQL condition in rule):

```
-- ISTRulesDataValidation.Error (condition)
```

(isnull(Age,0) < 0 OR isnull(Age,0) > 120)
Cross-record uniqueness:

```
exists(
 select 1
 from dbo.Members m
 where m.Key1 = d.Key1 and m.Key2 = d.Key2 and m.ID <> d.ID
)
```

Use alias d to refer to current row where supported.

8. System Tokens & Special Functions

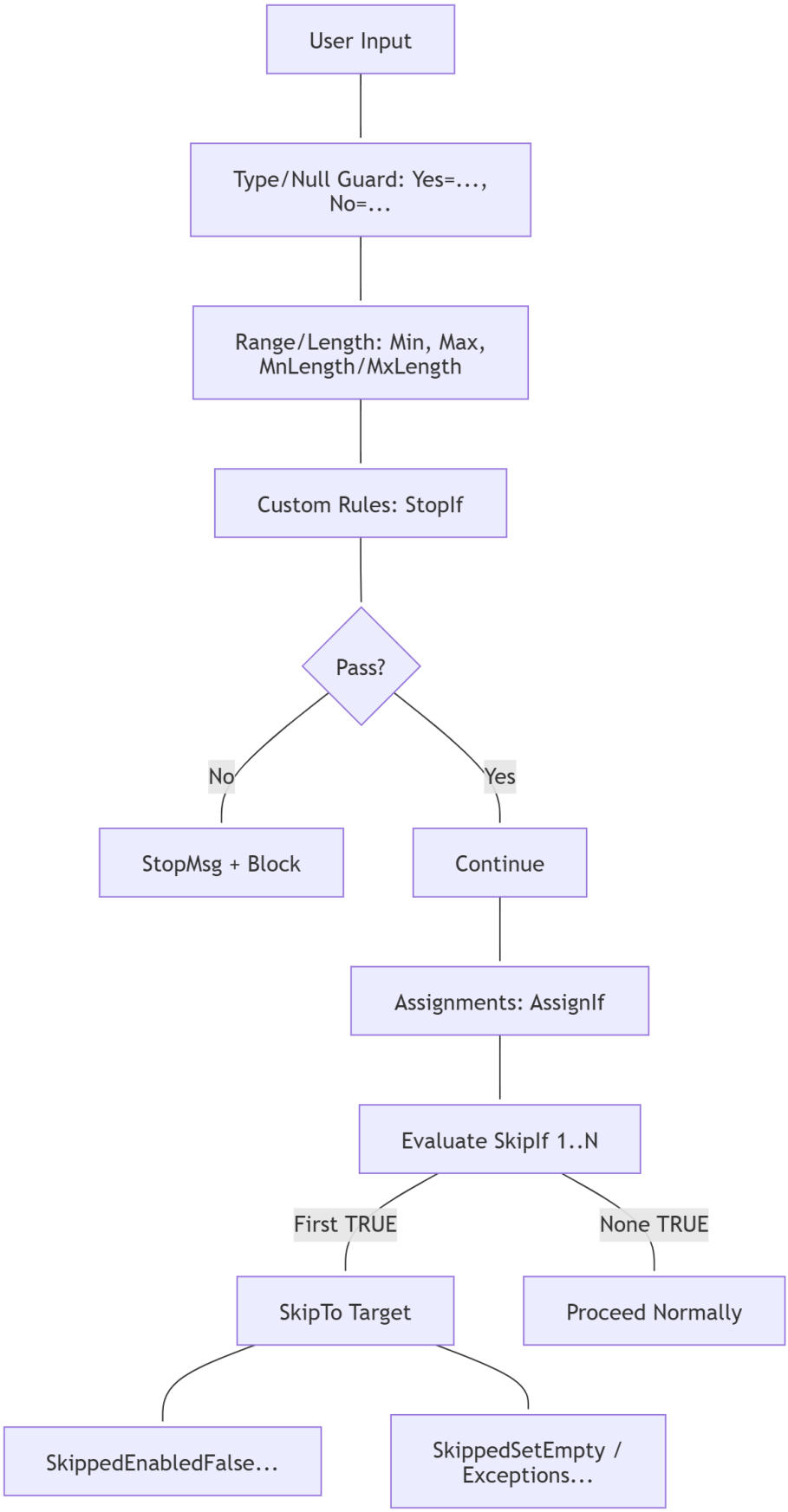
MOD11 checksum:

```
#{
 StopIf=#MOD11{RegNumber,8}=0;
 StopMsg='Registration number failed MOD11 check';
}
```

Common SQL helpers: isnull(), cast(), len(), exists, not exists, in, like, isDate().

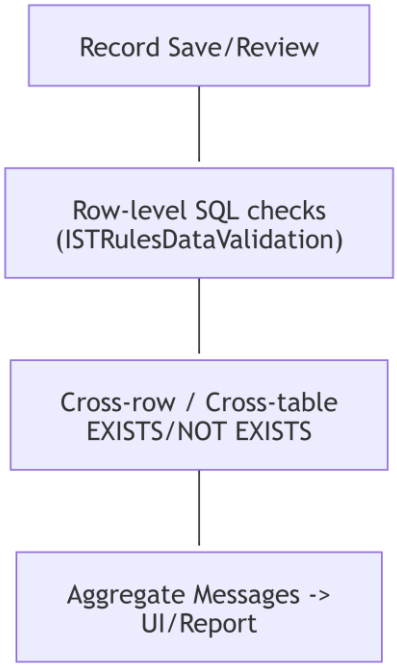
9. Diagrams (Flow & Lifecycle)

9.1 Field-Level Validation Lifecycle and Skip Resolution Order



```
flowchart TD
A[User Input] --> B[Type/Null Guard: Yes=...,No=...]
B --> C[Range/Length: Min, Max, MnLength/MxLength]
C --> D[Custom Rules: StopIf]
D -->{Pass?}
E -->|No| F[StopMsg + Block]
E -->|Yes| G[Continue]
G --> H[Assignments: AssignIf]
H --> [Evaluate SkipIf 1..N]
I -->|First TRUE| T[SkipTo Target]
T --> X1[SkippedEnabledFalse...]
T --> X2[SkippedSetEmpty / Exceptions...]
I  -->|None TRUE| N[Proceed Normally]
```

9.2 Row level validation (on Save)




```
flowchart TD
  A["Record Save/Review"] --> B["Row-level SQL checks (ISTRulesDataValidation)"]
  B --> C["Cross-row / Cross-table EXISTS/NOT EXISTS"]
  C --> D["Aggregate Messages -> UI/Report"]
```

10. Troubleshooting & Patterns

- Rule not firing? Confirm the event context (field vs. save) and field/alias names.
- Visibility conflicts: avoid mixing VisibleTrueIf and VisibleFalseIf on the same target.
- Cleared data surprises: when using SkippedSetEmpty, include required exceptions.
- Multi-response parsing: use clear delimiters to avoid substring collisions in like checks.
- Checksum failures: verify length/format (strip separators) before #MOD11{}.

Appendix 3: Copy-Paste Snippets

1. Professional Range & Type Check (Age)

```
#{
 Min=0; Max=120; Yes=numeric;
 StopIf=isnull(Age,0)<0 or isnull(Age,0)>120;
 StopMsg='Age must be between 0 and 120';
}
```

2. Conditional Required

```
#{
 RequiredIf=Status='Active';
 StopIf=isnull(ActiveDate,'')='';
 StopMsg='Active Date is required when Status is Active';
}
```

3. Multi-Response Bounds

```
#{
 Yes=MultiResponseNumeric; Min=1; Max=10;
 MnResponseCount=1; MxResponseCount=3;
 StopIf=ResponseCount<1 or ResponseCount>3;
 StopMsg='Select between 1 and 3 options';
}
```

4. Visibility (Show "OtherDetails" when "Other" is chosen)

```
#{
 VisibleTrue=OtherDetails;
 VisibleTrueIf=Option='Other';
}
```

5. Assignment (Initialize Status)

```
#{
 AssignTo=Status;
 AssignWhat='NEW';
 AssignIf=isnull(Status,'')='';
}
```

6. Batch Rule (SQL condition example)

```
-- In ISTRulesDataValidation.Error
(isnull(Age,0) < 0 OR isnull(Age,0) > 120)
```

Appendix 4 – Developer Q&A (Frequently Asked Questions)

This appendix collects frequently asked questions (FAQ) by IST developers, grouped by topic to provide fast, practical guidance during application development.

1. Metadata & Application Setup

- Q: How do I quickly create a new application using metadata from an existing one?

A: Use the `additionalParameters` field in _IST to define inheritance: `IST=OldAppCode` copies all metadata. You can override specific tables via `ISTTables=...`.

- Q: I added a new variable but it doesn’t appear on the form - what did I miss?

A: Ensure it exists in _ISTTableColumns, is valid for the selected time point, and isn't hidden via columnAttributes.

2. Logical Control & Events

- Q: How can I disable a field unless another field has a specific value?

A: Use: `#{EnabledFalse=fieldX; EnabledFalseIf=isnull(d.fieldY,'')<>'expectedValue';}`

- Q: What’s the difference between SkipTo, StopIf, and ExecMsgIf?

A: SkipTo moves focus. StopIf prevents saving. ExecMsgIf triggers a stored procedure with a message.

- Q: Can I reuse skip/stop logic across forms?

A: Yes, use metadata inheritance.

3. Virtual Fields

- Q: When does a virtual field recalculate?

A: By default, on each OnValidating event. Use :PK or :CalculateOnExit to control timing.

- Q: Can a virtual field reference another virtual field?

A: Yes, but avoid circular references. These are not automatically detected.

4. Reports & Output

- Q: How do I add a report with dynamic year and month filters?

A: Use placeholders like GGG and MMM in the SQL or header files. IST replaces them automatically.

- Q: How can I run a macro after the report is exported?

A: Set the macroExcel field to a .xlsm file with a macro of the same name in _ISTReportsProcedures.

5. Debugging & Testing

- Q: How can I preview a form without affecting production data?

A: Create a test appCode, assign it a separate test database, and test in a development environment.

- Q: I get no errors but nothing appears on the form - what can I do?

A: Check: ValidFrom/To, schema name, and tableOrder (0 hides table).

6. Validation & Logs

- Q: How do I find out what validation rules ran?

A: Check _ISTLogBatchLC in the DEPO database for executed validation entries.

- Q: How can I simulate a batch validation without real data?

A: Insert test rows into your test DB and manually run the validation module.

7. Security & Access

- Q: Can I restrict visibility of reports based on user roles?

A: Yes, use the `visibility` field in _ISTReportsProcedures with conditional SQL (e.g. f_CheckRole()).

- Q: Can I limit which applications users see in the IST dropdown?

A: Yes, using SQL views or AD role filtering logic (custom implementation).

8. Web & Mobile

- Q: Which metadata features are not supported in the Web or Android interpreter?

A: Some controls or complex layouts may render differently. Always test in web/mobile environment.

- Q: Can I deploy the same app across desktop and mobile?

A: Yes. All interpreters use the same metadata. Add overrides only if needed via SaveExecOnLoadParameters field in _IST metadata table.

9. Performance

- Q: How can I speed up forms with lots of virtual fields?

A: Use :CalculateOnExit=Field to limit recalculation. Avoid deeply nested queries in expressions.

- Q: Batch validation takes too long - what can I optimize?

A: Simplify WHERE clauses in validation rules and use indexed fields where possible.