

Traverse SDK

The Traverse version 11 Software Developer's Kit (SDK) is online, and available wherever and whenever you need it. Use the Traverse SDK to ensure that your modifications adhere to Traverse standards—making them easy to maintain, update, and upgrade.

The Traverse SDK includes:

- Framework Documentation and Programming Guidelines
- Schema Definitions
- Entity Relationship Diagrams
- Code Generator
- Object Documentation
- TSMX Builder
- Client Update Utility
- Task Panes, including a configuration file and a short tutorial

The screenshot displays the Traverse SDK documentation interface. On the left is a 'Table of Contents' pane with a tree view containing categories like 'Standards & Guides', 'TRaverse Framework', 'Business Layer', 'Data Constraints', 'Hints', 'Naming Guidelines', 'Others', 'Process-based Tasks', 'Regular Expressions', 'Reports', 'User Interface', 'Programming Model', and 'TRaverse Namespace'. The main content area is titled 'AddValidationRules' and contains the following text:

Validation rules that define business logic of the entity should be maintained at the business layer. This helps reduce the scattering of "entity knowledge" in various layers of software project.

Everstar allows creating lists of validation rules by defining "Common Rules" and "Entity Rules". Every entity class implements some sort of validation list. A simple example would be one that checks for a null value in a field. The best aspect of the Everstar implementation is the extensibility and flexibility that is achieved using delegates. These concepts should become more clear with the following discussion and examples.

The first step in implementing Entity level validation rules is to include an override for the base Entity "AddValidationRules" method. The override should include a call to the base method to insure that the validations of the base Entity are included in the list of validation rules. Once this method has been implemented, additional validations can be added to the entity by issuing the "AddRule" method of the "ValidationRules" list.

```
//implements the AddValidationRules method
protected override void AddValidationRules()
{
    //adds any base Entity validations to the list of validation rules
    base.AddValidationRules();

    //TODO - add validations
}
```

Adding a simple validation rule via the "AddRule" method requires nothing more than a "ValidationRuleHandler" and the name of a property to apply the validation too. The following example prevents the given property from being set to a null.

```
//prevent the property from being null
this.ValidationRules.AddRule(Validation.CommonRules.NotNull, "PropertyName");
```

More complex validations may require a set of validation rule arguments (ValidationRuleArgs) to provide additional information needed to process the validation. For those handlers that have specific requirements, a unique validation rule arguments declaration will exist to define the additional information that is needed. The following example uses the "StringLength" handler along with the "MaxLengthRuleArgs" to define both the name of the property to validate as well as the maximum length parameter value.

```
//limit the length of the property to 10 characters
this.ValidationRules.AddRule(Validation.CommonRules.StringMaxLength,
    new Validation.CommonRules.MaxLengthRuleArgs("PropertyName", 10));
```

Additionally, there may be instances where the specifics of the validation need to vary depending upon some other condition. To address this, each set of validation rule arguments can be given a conditional rule handler. The conditional rule handler is implemented locally within the Entity to determine if a specific condition exists. Any number of local methods can be created for identifying given conditions as long as they have a signature that matches the declaration of a ConditionalRuleHandler. Note that the signature includes a property name that can be used to process multiple conditions within a single method. The following example addresses a case where a date value must be within a given range whenever it is not null.

Traverse version 11 SDK Tools and Documents

Framework Documentation and Programming Guidelines

The framework provides mapping from relational databases to the object world, which allows application developers to focus on requirements and features rather than dealing with complexities of databases and SQL statements.

Schema Definitions

Online, searchable access to the schema definitions for the enumerated field values, system databases, and company databases is available for every application.

Entity Relationship Diagrams

This file contains the relationship diagrams for the version 11 Traverse schema. The two files are in PDF format, and the file names include references to the application information contained in each file.

Code Generator

This tool can be used to automate the creation of class assemblies that are compatible with the Traverse version 11 framework to support new schema entities. A 'Quick Start' guide to using the code generator is included in the file.

Object Documentation

These XML files facilitate the understanding of the methods available in the application. When the files are extracted into the folder that contains the Traverse programs (dlls) on machines that will be used for development, the Visual Studio Object Browser will show any extended comments that have been created in the source projects.

TSMX Builder

The TSMX Builder utility is used to create the SQL scripts that perform desired tasks on a database to accommodate customizations. These can include the creation or alteration of database objects, the manipulation of data, or virtually any other SQL compatible instruction. Once these sets of instructions have been built and tested individually, they can be compiled into an update package using this utility.

Client Update Utility

The Client Update Utility is an extension of programs available in Traverse that automatically checks for updated files deployed to a server. This tool has a dual purpose. First, as a development tool, it can be used to build the AssemblyListCustom.xml file for custom files that should be deployed to client installations. Second, when deployed into the installation environment, this tool can automate checking and updating client files using external tools such as Windows Task Scheduler. This can also be run as part of a logon or batch process for users. Updates could be silently loaded prior to the loading of Traverse, so the user does not receive a message regarding the availability of new updates after the software has loaded.

Task Panes

This file includes the current OSI-provided Task Panes for Traverse, a sample configuration file, and a short tutorial on configuring your system to use Task Panes.