


# Web Development Framework

 empower Web Development Framework (“WDF”) is a Software Development Toolkit (“SDK”) used to build empower Entities (APIs and Web Services) that can be consumed by the empower Entity Management Portal (Web User Interface) or other 3<sup>rd</sup> Party Applications. The empower Entities update the Dynamics SL database, applying all necessary business logic and ensuring valid maintenance of Dynamics SL records.

An empower Entity is analogous to a Dynamics SL screen. It is an API, which can be published as Web Service. It defines the database tables to be updated, controls field behaviour and describes all business logic for the Entity.

The WDF has a number of key components:

<b>Schema Editor</b>	Defines the tables, fields and field properties for the Entity.
<b>Logical Date Types</b>	A collection of fields and business logic required to control field behaviour and interaction. This can be as simple as a single text field, or a complex function such as Currency handling.
<b>Enumerations</b>	Small data tables for managing field value lists. Eg Statuses.
<b>Code Generator</b>	Using the Schema files and LDTs the Code Generator generates the base generalised source code for the Entity. Manual code can then be injected into the Entity prior to publishing.
<b>Web Form Designer</b>	Builds/Edits the empower Web Forms.
<b>Web View Designer</b>	Builds/Edits the empower Web Data Lists (“Views”).
<b>System Manager</b>	Allows definition of module or role tailored menus, controls users, groups, security and general empower configuration.

## Schema Editor

The Schema Editor enables the user to define the tables and fields that will be available in the Entity. It defines all the properties for the fields including whether they are read only or writeable, their type (Text, Integer, etc.), value defaults or limits and so on. Field properties can be assigned and controlled by LDTs.

There are two principle ways to build an empower Entity.

- ✚ Importing existing Dynamics SL tables and applying roles / LDTs to all fields.
- ✚ Building an entirely new Entity based on LDTs and custom user definitions.

### IMPORTING EXISTING DYNAMICS SL SQL TABLES AND FIELDS

The Entity Builder allow users to select an existing Dynamics SL table and import all fields and their SQL column properties. The Schema Editor is used to assign the properties for all fields within the Entity. There are a number of tools and functions provided to auto-assign field properties. Users can then manually edit properties for all fields defining custom behaviour where required.

**Roles, LDTs** Each Entity Field must have a Role and Logical Data Type “LDT” assigned.

The LDT defines the field properties and business logic to be generated for the field when the Entity Code Generator is run. LDTs are the base building blocks for all Entities. LDTs can be built or modified by users.

Example: All fields with ‘Acct’ in their column name are assigned the Role and LDT for a GL Account. This sets the Foreign Key join to the GL Account Entity and describes all the Business Logic for the GL Account field.

Using Roles and LDTs ensures every empower Entity will apply exactly the same field properties and business logic throughout the application. If the business logic is changed in an LDT, this will be automatically applied to every instance of the LDT in every Entity throughout the empower application.

**Enumerations** In Dynamics SL, some fields have a dropdown list of options. This is common for fields like Status or minor option settings. Dynamics SL stores the list data in the SL screen. Latest development architecture stores these value lists in Enumerations “Enums”. A single Enum can then be re-used in multiple fields throughout the application.

The empower WDF provides a Enumeration Editor to add/edit value lists.

An “SL Assembly Tool” is also provided, which reads the field value lists from the Dynamics SL screens and builds the necessary Enums required for Dynamics SL Entities.

**Foreign Keys** Many fields in Dynamics SL are validated against data in another SL table. SL uses PV lists for value selection and validation. empower uses a true Foreign Key join to the related Entity, validating entered data against the records in the related Entity. A ‘Look-up’ View is used to apply filters where a narrower data set is required, such as active only.

**Schema Tools** The empower WDF provides functions to build Enumerations and assign, Roles, LDTs and Properties to all Entity Fields based on their values within standard SL screens.

A User can build an Entity based on standard Dynamics SL screens within only a few minutes. By importing the underlying tables, running the Schema Tools to assign LDTs and field properties, then making manual adjustments, a User can quickly and easily build an Entity (Web Service) for almost any SL screen.

### BUILDING OR EDITING AN ENTITY FROM LOGICAL DATA TYPES

Where a new Entity is required, that does not have in existing table in the Database, the user can start with a blank schema. A list of all Logical Data Types is provided and these can be selected and copied to the Entity Schema. It is also possible to copy an LDT into an existing SL Entity and extend the SL tables.

Coping an LDT into an Entity will create all the necessary fields, assign the field properties, and build the business logic described by the LDT. New Entities can be built using standardised building blocks (LDTs) ensuring consistent field naming, field property definition and code generation.

## Windows Schema Editor

empower provides both a Web Schema Editor and a Windows Schema Editor. The Windows version includes functions for auto-assigning field properties and defaults, and for validating properties against known Dynamics SL rules and templates to ensure Dynamics SL conventions are being correctly applied.

Member Name	Logical Type	Accessors	Comp	Related Database Column	SQL Type	True Text	False Text	Mask Type	Mask Length	Decimal Places	Min Value	Max Value	Default Type	Default Value	Control Type	Visible	Enabled	Required
AccrRevAcct	IAccount	{ get; set; }	<input type="checkbox"/>	Customer.AccrRevAcct	char(10)										Text Field	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
AccrRevSub	ISubAcct	{ get; set; }	<input type="checkbox"/>	Customer.AccrRevSub	char(24)										Text Field	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Address	IAddress	{ get; }	<input checked="" type="checkbox"/>													<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
-- Address.Addr1	String	{ get; set; }	<input type="checkbox"/>	Customer.Addr1	char(60)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
-- Address.Addr2	String	{ get; set; }	<input type="checkbox"/>	Customer.Addr2	char(60)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
-- Address.Attn	String	{ get; set; }	<input type="checkbox"/>	Customer.Attn	char(30)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
-- Address.City	String	{ get; set; }	<input type="checkbox"/>	Customer.City	char(30)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
-- Address.Country	String	{ get; set; }	<input type="checkbox"/>	Customer.Country	char(3)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
-- Address.Email	String	{ get; set; }	<input type="checkbox"/>	Customer.EMalAddr	char(80)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
-- Address.Fax	String	{ get; set; }	<input type="checkbox"/>	Customer.Fax	char(30)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
-- Address.Name	String	{ get; set; }	<input type="checkbox"/>													<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
-- Address.Phone	String	{ get; set; }	<input type="checkbox"/>	Customer.Phone	char(30)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
-- Address.Salut	String	{ get; set; }	<input type="checkbox"/>	Customer.Salut	char(30)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
-- Address.State	String	{ get; set; }	<input type="checkbox"/>	Customer.State	char(3)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
-- Address.Zip	String	{ get; set; }	<input type="checkbox"/>	Customer.Zip	char(10)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ApplFinChrg	Boolean	{ get; set; }	<input type="checkbox"/>	Customer.ApplFinChrg	smallint	1	0						Const...	true	Check Box	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ArAcct	IAccount	{ get; set; }	<input type="checkbox"/>	Customer.ArAcct	char(10)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
ArSub	ISubAcct	{ get; set; }	<input type="checkbox"/>	Customer.ArSub	char(24)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
AutoApply	Boolean	{ get; set; }	<input type="checkbox"/>	Customer.AutoApply	smallint	1	0						Single...	IARSetu...	Check Box	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
BillingAddress	IAddress	{ get; }	<input checked="" type="checkbox"/>													<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
-- BillingAddress.Addr1	String	{ get; set; }	<input type="checkbox"/>	Customer.BillAddr1	char(60)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
-- BillingAddress.Addr2	String	{ get; set; }	<input type="checkbox"/>	Customer.BillAddr2	char(60)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
-- BillingAddress.Attn	String	{ get; set; }	<input type="checkbox"/>	Customer.BillAttn	char(30)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
-- BillingAddress.Civ	String	{ get; set; }	<input type="checkbox"/>	Customer.BillCiv	char(30)										Text Field	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

The steps for creating an Dynamics SL Entity would be:

1. Select 'Add Table' from the top line menu.

Link to the Dynamics SL Application Database and select the primary table (Eg Customer). This imports the SQL schema into the Schema Editor setting base database field properties, SQL Data Type, etc.

2. Run the 'Load Assembly' function.

This function applies predefined SL rules to set the Logical Data Type ("LDT"). As examples, it applies the IAddress LDT to all address fields, IAudit to all Crtd\_ and Lupd\_ fields, IAccount to all Acct fields. The LDT sets many field properties and the Assembly updates many more including default values.

3. Manually adjust any Schema properties and Enumerations, where necessary.

4. Run the Code Generator to build the source files which can be opened in Visual Studio.

Source code files are built based on standardised business logic inherent in the LDTs.

5. Inject Manual Code where desired to define custom business logic for the specific Entity.

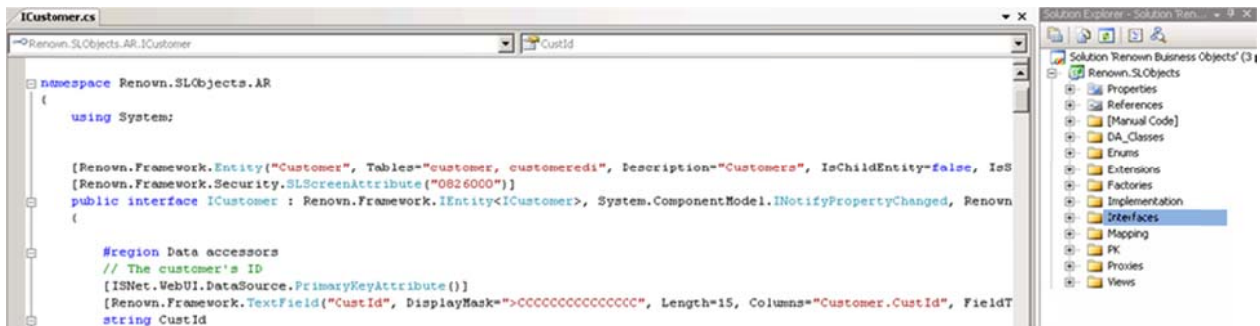
6. Compile and Publish the Entity .

Generates the DLLs, APIs and Web Services used by the Enterprise Management Portal and/or other 3<sup>rd</sup> Party applications. It also builds a default Web Form and Web Views in the empower Portal.

7. In empower Portal add or modify the Web Forms and Views and define User Access to the Entity.

## Code Generator

The empower WDF Code Generator reads the Schema file and using the Logical Data Types builds the source code files required by the Entity. Each time the empower Code Generator is run it will drop and re-create these source code files, with the exception of the "Manual Code". Manual Code is used for custom, non generalised code required by the Entity, injected by Renown, Patterns or Clients.



- Manual Code** An empty place holder is created for manually defined source code. This is used to inject custom code into the Entity. Not all business logic can be standardised into LDTs. Some custom code may be need to control field behaviour and functionality for an Entity. The Manual Code allows Renown and Users to inject code using similar concepts to Events, in the traditional Dynamics SL VBA environment.
- DA Classes** empower uses the NHibernate framework to implement object-relational mapping. empower creates a data access object for all database tables. empower Entities can be mapped to multiple tables, or data access objects. `ICustomer` links to `dbo.Customer` and `dbo.CustomerEDI` using `Customer` and `CustomerEDI` data access objects.
- Enumerations** Enumerations define all the data lists used in field drop-downs. This does not include PV lists from other Entities, but small combo box lists normally maintained in an SL screen.
- Extensions** Following best practice patterns, extensions are used for non-core generalised methods.
- Factories** The Factories create, load and instantiate the Entities. They are the entry point for 3<sup>rd</sup> Party development created outside the empower Framework.
- Implementation** The C# classes that implement the business logic interfaces. This generated code can be extended via partial classes in the Manual Code section.
- Interfaces** An Interface is created for each Entity, Eg `ICustomer`. The interface describes the programmatically behaviour that developers must conform to when consuming the API. It defines the methods and classes available to the programmer for the Entity.
- Data Mapping** An XML mapping file is generated for every data access object. It is used by NHibernate to bind the database schema to the .NET objects. It links the Dynamics SL tables to the related empower Entity.
- Primary Keys** Other classes may be generated such as Primary Keys which describe the primary keys used by the Entity in the database.
- Proxies** Proxies are a light weight object returning the field list and properties for the Entity, but no data. This improves performance when loading an Entity. All related Entities, potential loaded due to foreign key associations, load only the proxy instead of their full Entity.
- Views** Views provide a flattened representation of the class hierarchy and convert property types into primitive .Net types like `String` and `Int32`. Views facilitate Web User Interfaces by providing a flat structure of the Entity Schema.

## Web Forms and Web Form Designer

### WEB FORMS AND USER SECURITY

A Web Form is the User Interface for data entry into the Entity. empower will provide pre-built Web Forms for all Dynamics SL screens. The standard forms, shipped with the empower installation, can be copied and or modified by Partners or Customers, using the Web Form Editor.

Access Rights are granted to Entities, Web Views and Web Forms controlling user access at three levels:

**Entity Level** Users can be granted Create, Read, Update and Delete for an Entity. Specific fields within the Entity can be set to disabled or not visible if required.

**Record Level Views** Views can be defined, with filters on multiple fields, which limit the records that can be retrieved from the database and displayed to the user.

**Field Level Views** Views can be defined, with filters on multiple fields, which limit the records that can be retrieved from the database and displayed to the user.

**Forms** Forms can be created showing only specific fields available to specific groups.

User Groups, or Roles, will only be granted access to the Entities, Views and Forms they require.

Users launch the Web Forms by double clicking a record (row) in the Web Views (data lists / grids). The user can select which Form they wish to launch for the Entity prior to double clicking the record.

### WEB FORM DESIGNER

The empower Web Form Designer lets Partners and Customers modify the Renown provided Forms or create new Web Forms. Each Form inherits the fields, field properties and controls, from the base Entity. Where Entity properties are fixed they cannot be modified in the Form. All other field properties can be modified. The Forms Designer is analogous to the Dynamics SL Customisation Manager for a screen.

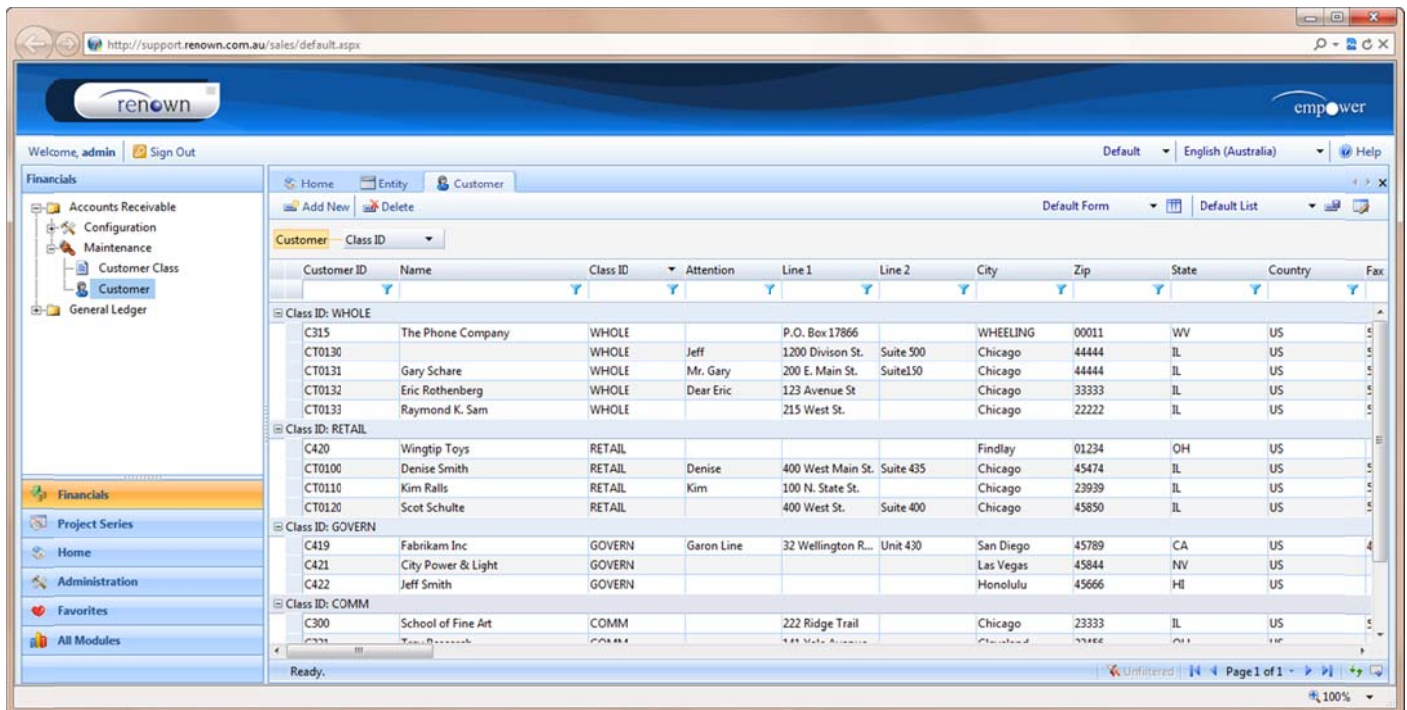
The screenshot displays the Web Form Designer interface. The main window shows a form titled 'Main' with a 'Caption' of 'Main' and 'Page Order' of '1'. The form contains several fields: 'Customer ID' (text), 'Status' (dropdown), 'Class ID' (text), 'Name' (text), 'Credit and Pric..' (text), 'Trade Discount ..' (text), 'Limit' (text), 'PriceClassID' (text), 'Terms' (text, highlighted in yellow), 'Credit Control' (text), 'Credit Check' (dropdown), 'Grace Period' (text), and 'Credit Manager ..' (text). The 'Available Fields' panel on the left lists various fields like 'A/R Sales Accou..', 'Accrued Revenue..', 'Buyer ID', etc. The 'Properties' panel on the right shows the 'Terms' field's properties, including 'Object Name', 'Type', 'Control Type', 'Data Type', 'Datasource', 'Data Text Field', 'Data Value Field', 'List Items', 'Async Validate', 'Is Identity', 'Is Primary', 'Is Require', 'Behaviour', 'Is Read Only', 'Is Visible', 'Appearance', 'Caption', 'Display Column', 'Display FK Label', 'Default Value', 'Display Format', 'Edit Format', 'Highlight Type', 'Mask Expression', 'Skip Optional', and 'Show Editor'.

Forms allow multiple Tabs and have a similar look to SL screens, albeit in a web UI. Entity fields are added to tabs using formats for radio choice, dropdown, look-up or simple text, numeric and date entry. Headers can be used to separate and group logical collections of fields similar to SL frames.

## Views Designer

The empower Views Designer lets Partners and Customers modify the Renown provided Views or create new Web Views. A view is a data list or grid of data, that can be grouped, sorted and filtered in any way.

User Groups are granted access to specific Views so Users can be restricted to seeing and/or updating only specific fields and filtered records that they have access to.



**Columns** Select the specific columns (fields) you wish to display in the View by checking the row.

**Filters** Add filters to multiple columns, setting operands for equals, like, contains, etc.

**Sort / Group** Drag multiple column headers into the 'Group By' panel to define tiered grouping. Sort by any column in the list by clicking on the column header.

**Read/Write** Define if the view and/or individual fields in the view are read only or can write back to SL.

