

## MVC Overview

MVC is part of a new generation of application development and integration tools from Hyfinity. Based on native XML, MVC enables the rapid development of enterprise-scale browser-based applications. MVC provides intuitive graphical IDEs for rapidly assembling complete interactive applications. One of the key differences between MVC and more traditional approaches is the information flow concept that preserves document-based information from capture, validation through to data binding and routing. All of this is performed using native XML. The key features of MVC include:

- Web Based Graphical IDEs
- Schema Based Development
- Auto Gen Validation Logic
- Automatic Data Binding
- Automatic Pre-Population
- Dynamic, Content Based Page Delivery

MVC uses XStudio for developing XML applications and XPlatform for hosting the resulting applications. MVC is primarily concerned with the development of interactive self-service applications and employs a tool within XStudio known as FormMaker for building complete web applications.

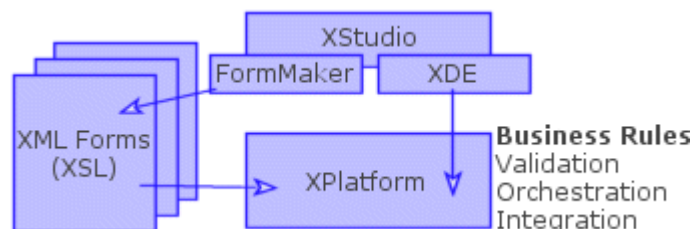


Figure 1. MVC Design and Deployment Architecture

An additional MVC component is present within XPlatform known as SXForms. This is a custom engine for delivery and post-processing of XML forms. Additional information is provided in the following sections.

### Logical architecture

Logically, MVC applications can be viewed as having the capability of editing XML documents in-place. This provides the ability to render an XML document as a form on the browser. The captured information is then validated and mapped back to an XML document.

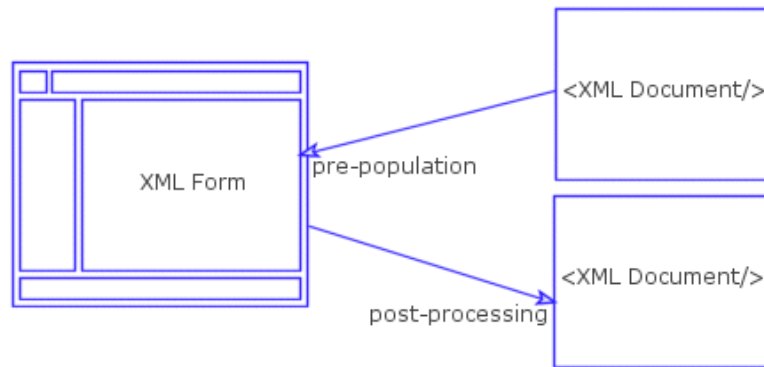


Figure 2. MVC in-place XML editing concept

## Physical architecture

Physically, browsers do not support XML forms in a uniform way. XHTML is still the primary language for browsers. MVC applications don't require any downloads on the browser, instead, a series of innovative steps are used to provide logical XML-based development to enable the in-place XML editing paradigm. To achieve this, the XML information is transformed to XHTML on the server using XSL. The information that is captured on the browser is posted and transformed from XHTML name-value pairs to XML on the server-side. This provides the ability to logically view the XML forms as editing an XML document in-place.

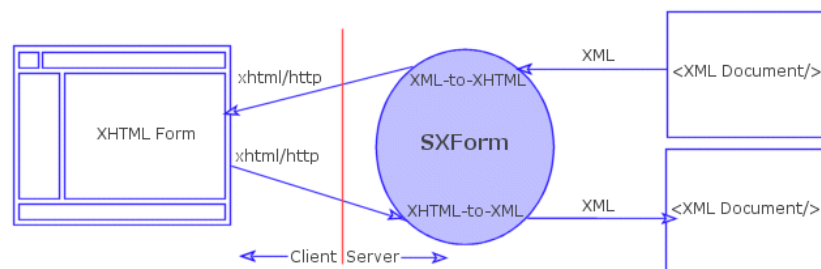


Figure 3. Automatic bi-directional data binding

## Schema-based development

Each form in an MVC application is derived from an XML-schema where possible. Unlike most technologies, MVC uses an information modelling approach to ensure better-engineering and cohesion between different elements of a distributed application. Rather than providing a form painter and starting with fields on a canvas, followed by defining their attributes, MVC prefers information to be defined, independent of the eventual display format of such information.

For example, if a person's surname, forename and age fields require capture then this information can be defined as an XML schema. The amount of detail in the schema will dictate the level of automation that will be achieved in the resulting application. For example, if all three fields are defined as alphanumeric then the information will simply be captured and submitted by default unless specific validation rules are explicitly specified. However, if the age field is defined as integer then various validation steps will be performed automatically. Further, if the age is defined as being between 18 and 60 for example, then scripting will be automatically generated and activated to ensure proper validation.

Schema-driven development requires a different approach to traditional 'form-painting' tools, providing many advantages.

- Well engineered applications, with good cohesion throughout the document flow route.
- Reuse of existing schemas.
- Once defined, schemas can be reused across multiple forms, providing massive reusability.
- Schemas can be used for purposes other than defining XML forms. For example, messaging.
- Provides a display-independent representation of form information.

## **Application architecture**

As well as the XML forms, each MVC application uses additional separation of key web application components to enable better engineering and easier future maintenance. Some components apply application-wide with page-specific overrides. All MVC applications require the following:

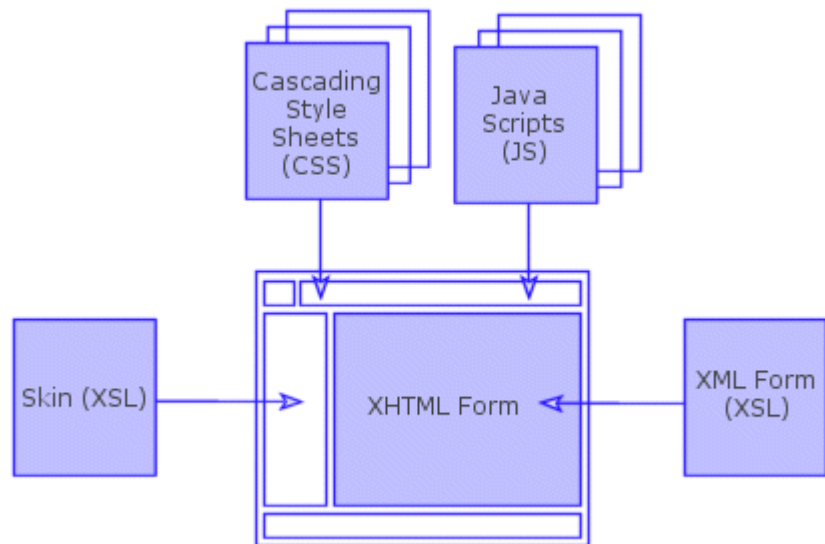


Figure 4. The anatomy of MVC Forms

- common Cascading Style Sheet - This is used to provide a generic application-wide look-and-feel to the application and provides separation of content and presentation.
- A common application Skin - This is in the form of an XSL and provides a template for the overall application. The skin will provide placeholders for menus, logos, headers, footers, etc. The actual content will override this template as necessary for each page, but the parts that are not overridden will remain the same.

The CSS and Skin require definition only once and provide an overall look-and-feel and layout for the web application. The main purpose of MVC however is to enable the creation of XML forms. This is achieved by using a component of MVC known as FormMaker. FormMaker is part of XStudio and provides a web-based IDE for creating XML forms. In addition to the CSS and Skin described above, FormMaker can be used to produce the following elements of a web application:

- XML forms encapsulated in XSL. These XSLs are transformed to XHTML during execution to enable wide browser coverage without downloads.
- Page-specific CSSs to override or complement the application-wide CSS
- Javascripts. A common javascript file is created for the application, which will contain generic scripting for client-side validation. A large part of this scripting file is generated and activated based on the schema information that is present for each form. Additional server-side logic can be included to compliment the client-side scripting.
- XML Data Mapping information. Similar to the scripting, this information is distributed between the client and the server and enables the data to be mapped bi-directionally between the browser and the underlying XML documents. Once

defined, this element is automated and very transparent to the designer.

## Development steps

Traditionally, dynamic web applications need to compose XHTML for the browser and decompose the information posted from the browser in server-side logic. MVC uses innovative XML techniques to eliminate bulk of the effort required to pre-populate and validate information to and from the browser. MVC uses XPath to enable information to be mapped between the XML form and the underlying XML document from which the data is derived. This is a two-way mapping that provides pre-population of information to the browser and the subsequent capture after the information has been posted from the browser. This enables the automation of a large part of server-side scripting role, typically associated with web application development. The key steps for developing MVC applications include the following:

- Design screen navigation model for complex applications.
- Design or reuse a CSS and application Skin.
- For each screen, define or reuse a schema as a starting point.
- The generic application or specific page(s) can be generated at this stage for initial view. Subsequent steps enable individual fields or groups of fields to be selected, styled, positioned, validated and mapped.
- Indicate fields that will be part of each form.
- 'Fine-tune' display and validation for each field if required.
- Bind fields to XML elements in both directions using XPath.
- Design and adapt server-side nodes for page delivery and data-binding. All the above steps can be achieved using FormMaker.

The diagrams in the following pages lead through the main MVC screens used to build E-Forms. The first screen is the Application Map screen used to create the overall design of related sets of forms and pages. This screen is also used to setup the overall theme of the forms application, including any schema standards that may be applicable.

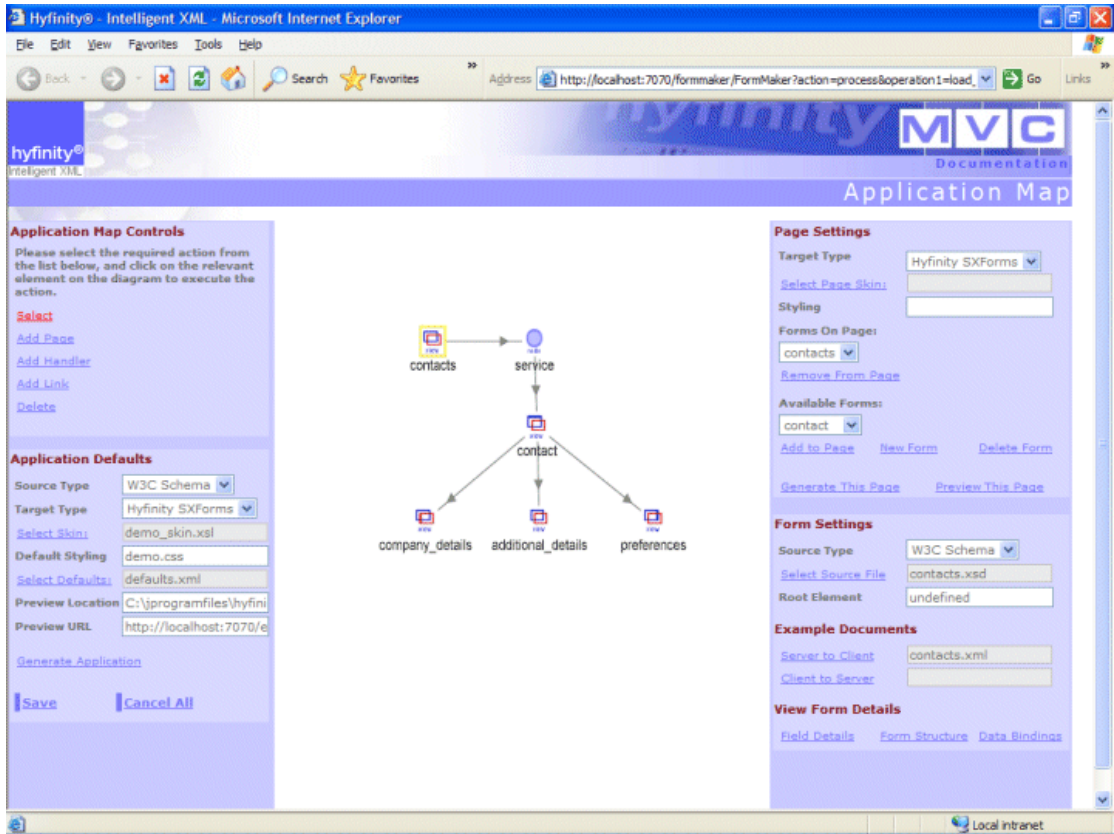


Figure 5. Application Map Screen. Form navigation and dependency.

For each form that is defined in the Application Map screen, a Form Structure screen is used to define groups of information containing individual fields. The information groups can contain nested repeating groups of information as appropriate. If schemas are used here then the information in the schemas can be dragged-and-dropped on to the form.

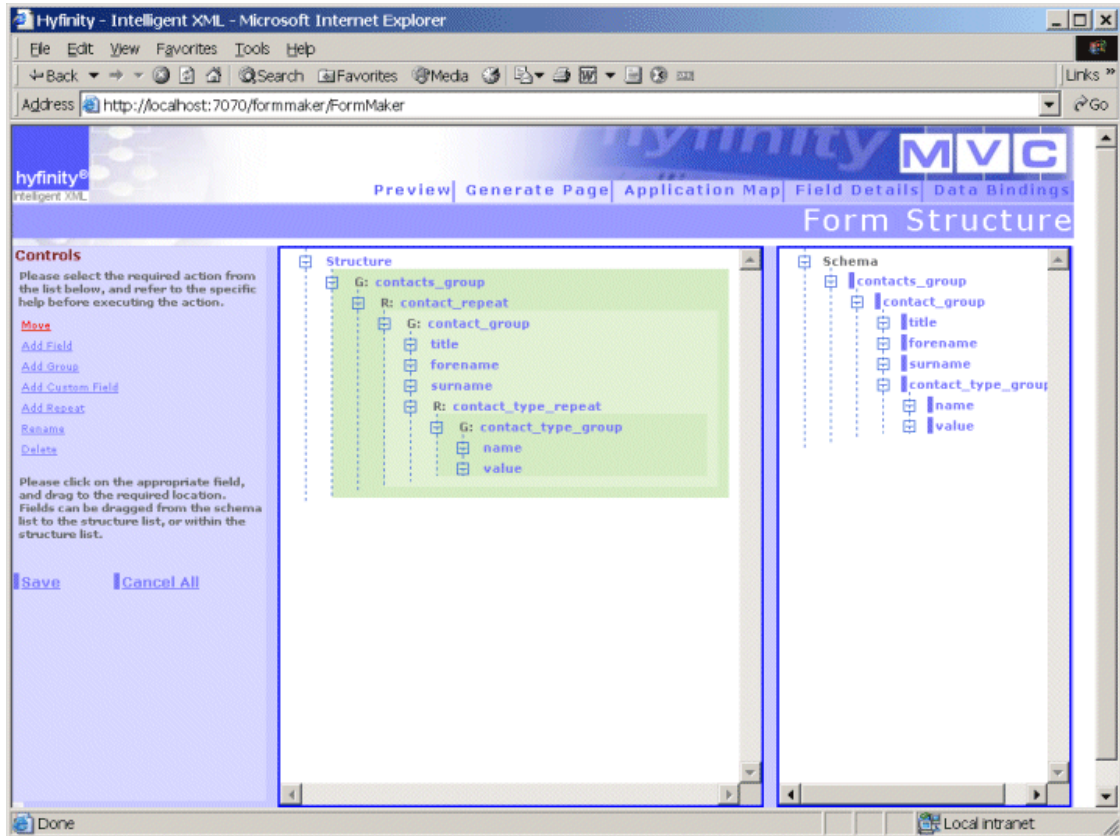


Figure 6. Form Structure Screen. Information structures within forms.

Once a form structure has been defined the form can be generated and previewed. Following this preview, each repeat, group or individual field can be selected and defined in more detail. For example, the type of control used to represent the field on the form, field labels, styling, etc. All this information will be initially defaulted using a combination of the CSS, application skin and the schema information if defined. This information can be changed or supplemented as required.

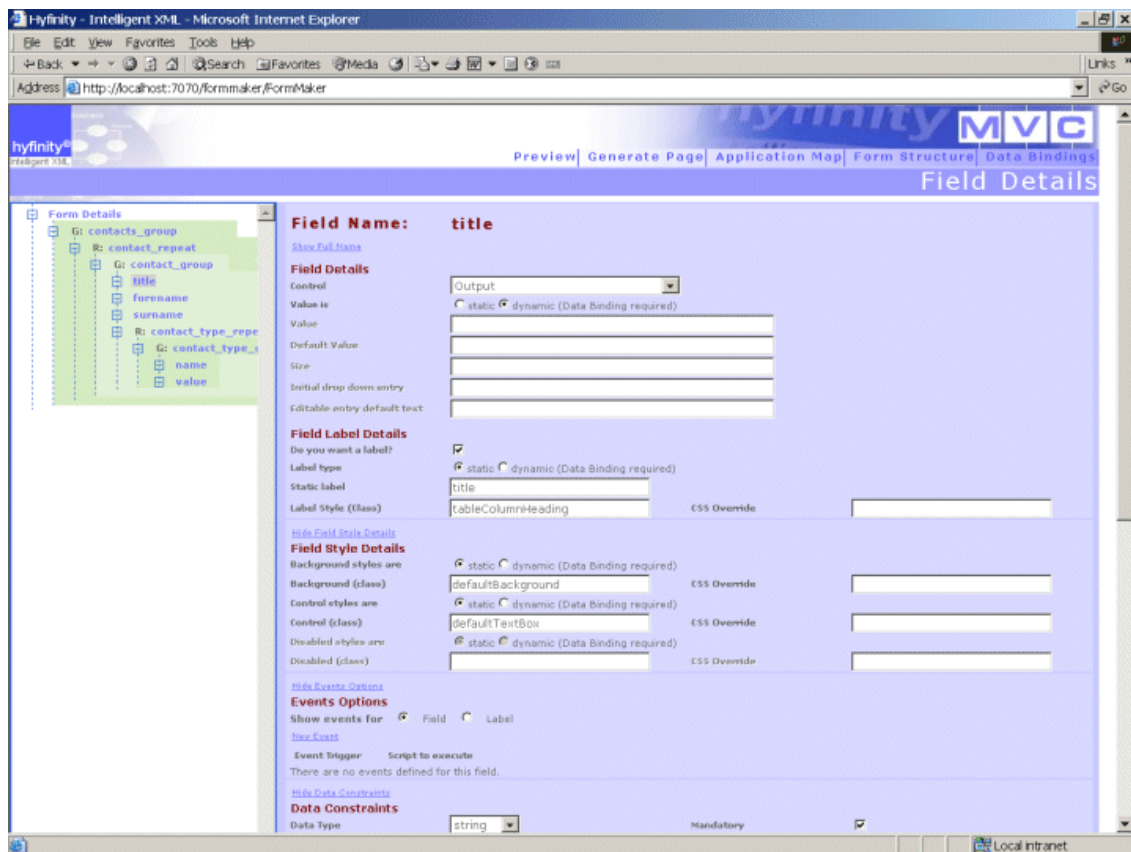


Figure 7. Field Details Screen. Fine-tuning individual form elements

Each field can have data constraints. If a schema was used then these constraints will be automatically generated and can be appended to or removed from manually if desired.

Hide Data Constraints  
**Data Constraints**

Data Type: string

Mandatory:

Enumerations

Data Value	Display Text
Mr	Mr
Mrs	Mrs
Ms	Ms
Dr	Dr
Sir	Sir

Up Down

Save Remove

Use these enumeration values for output display?

Min Inclusive:

Min Exclusive:

Min Length:

Length:

Pattern:

Max Inclusive:

Max Exclusive:

Max Length:

Figure 8. Data constraints

For certain field types there are sophisticated value formatting and conversion routines available. For example, dates can have different display and storage formats and these transformations can be automatically handled.

Hide Value Conversions  
**Value Conversions**

Date Formats

Data	Display
yyyy-MM-dd	dd/MM/yyyy

Case: Preserve

Whitespace: Preserve

Figure 9. Value conversions, including date manipulations.

Information on forms can be grouped for easier styling and manipulation. For example, complete groups of information may be hidden, highlighted, etc. based on other values on the form. There are also sophisticated features available for different display formats for information groups, such as horizontal, vertical, tabular, etc.

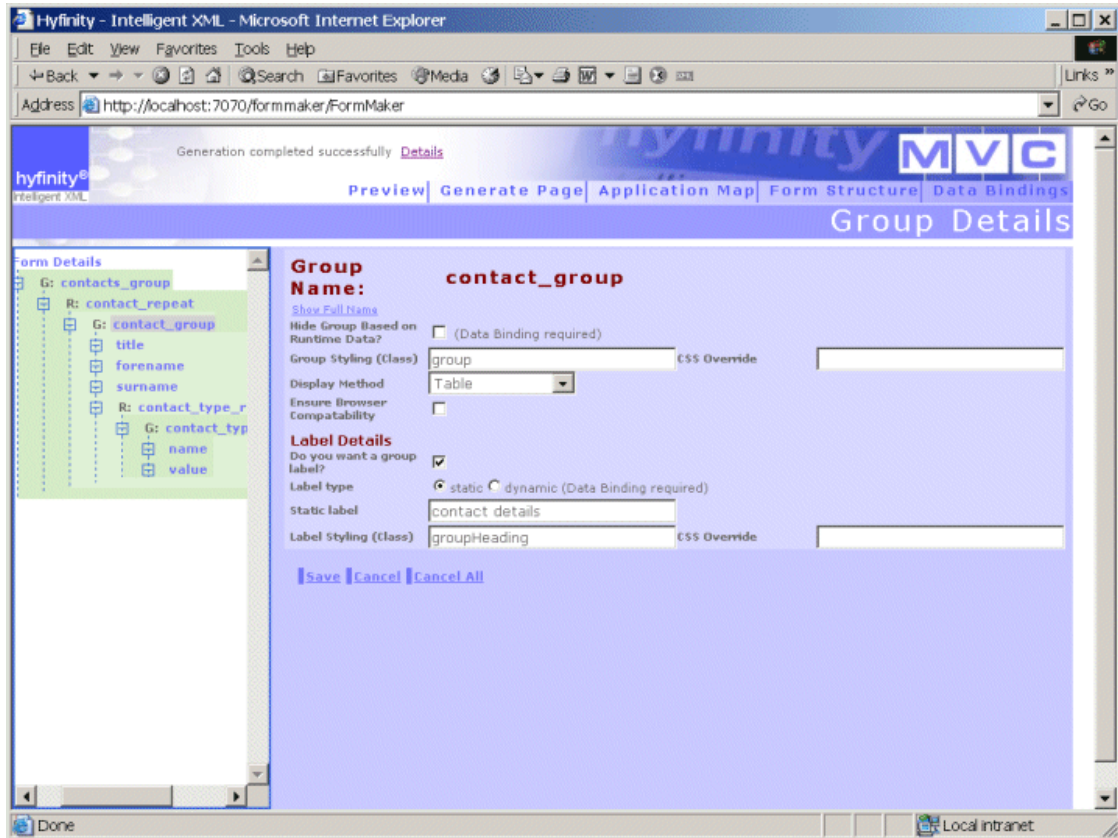


Figure 10. Grouping information on forms

Information groups can be repeated. There are various options available for the display and styling of such repeating groups of information, including sorting.

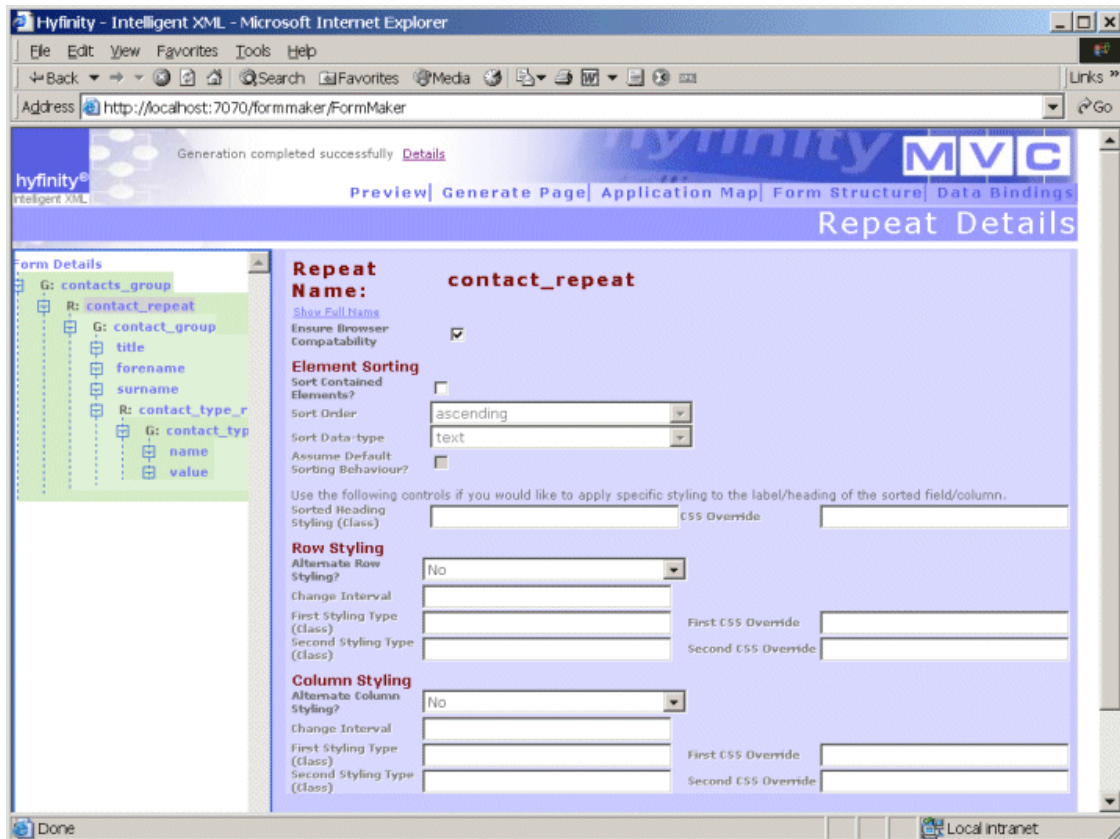


Figure 11. Repeating information groups in forms

The forms can be repeatedly previewed until the desired effect has been achieved. Once the individual repeats, groups and fields have been fine-tuned then each field can be mapped from a data source and back to a data target after completion. If XML instance documents were specified in the Application Map, then these can be used here to drag-and-drop fields onto the binding links. The binding information is automatically completed wherever possible.

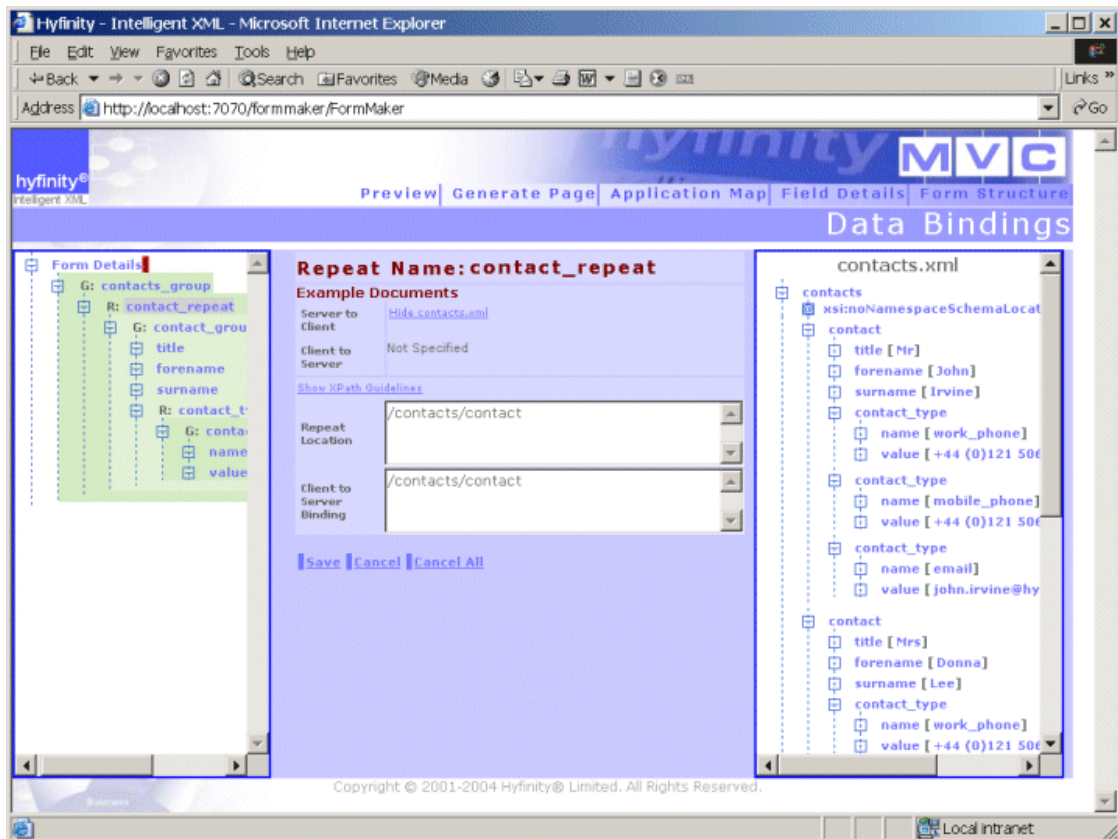


Figure 12. Automatic bi-directional data bindings screen