Architecture

Microsoft Dynamics CRM 4.0

Field-level Security in Microsoft Dynamics CRM: Options and Constraints

White Paper: "Nuts and Bolts" Series
Security and Authentication in Microsoft Dynamics CRM

Conceptual Application

Date: November 2009

Microsoft Dynamics™

Acknowledgements

Initiated by the Microsoft Dynamics CRM *Engineering for Enterprise* (MS CRM E^2) Team, this document was developed with support from across the organization and in direct collaboration with the following:

Key Contributor

Roger Gilchrist (*Microsoft*)

Technical Reviewers

Mahesh Hariharan (*Microsoft*)
Mahesh Vijayaraghavan (*Microsoft*)
Elliot Lewis (*Microsoft*)
David Jennaway (*Excitation UK*)

The MS CRM E^2 Team recognizes their efforts in helping to ensure delivery of an accurate and comprehensive technical resource in support of the broader CRM community.

MS CRM E^2 Contributors

Amir Jafri, Program Manager

Jim Toland, Content Project Manager

Feedback

Please send comments or suggestions about this document to the MS CRM E^2 Team feedback alias (entfeed@microsoft.com).

Table of Contents

Introduction 5
Controlling Access to the Dynamics CRM Platform 6
  Interaction Points 6
  Access Channels and Methods 7
  Usability Considerations 9
  Role- and Rule-Based Filtering 9
    Role-Based Filtering 9
    Rule-Based Filtering 10
Options for Filtering Data 11
  Access via Forms 11
    Consequences of Restricting Access by Using Client-Side JavaScript 11
    Restricting Access by Using Plug-Ins 11
  Access via Views/Advanced Find 13
  Access via the CRM SDK Web Service 13
  Access via the CRM Metadata Web Service 13
  Access via the Export-to-Excel Feature 13
  Access via Reports 15
  Direct Access to the Database 16
  Access via Outlook Synchronization 17
  Access via the Offline Client 17
Implementing Data Filters 18
  Plug-Ins 18
    Create 18
    Retrieve 18
      Pre-Retrieve Plug-In 19
      Post-Retrieve Plug-In 19
    Update 19
    Execute: FetchXML 20
      Pre-Execute 20
      Post-Execute 21
    RetrieveMultiple 23
      Pre-RetrieveMultiple 23
      Post-RetrieveMultiple 23
  The Export to Excel Feature 24
Performance and Scalability 25
Alternate Design Patterns 26
  Multiple Entities 26
    Characteristics 26
    Enhancements 27
  Augment on Retrieve 27
  Different Channels 28
Conclusion 29
Appendix A: Summary of Filtering Mechanisms 30

Preface

CRM E^2 Nuts and Bolts Series Overview

The MS CRM Engineering for Enterprise (E^2) *Nuts and Bolts* (NB) series is an expanding set of topical content, with each offering providing detailed information about the internal mechanisms related to a specific area of functionality within Microsoft Dynamics CRM 4.0.

NB Series offerings are designed to provide detailed technical resources that:

- Address often repeated queries to Technical aliases
- Consolidate answers, links, etc., that are generated in response to those queries
- Offer multiple levels of complementary information to support a broader, multi-perspective understanding of the topic
- Convey the baseline "principles" users require to begin to address related but tangential technical queries
- Present content using a consistent structure and "look and feel"

Audience

The target audience of the NB Series includes (but is not limited to):

- Solution Architects
- Application Architects
- Infrastructure Architects
- Consultants
- Developers

NB Article Content and Structure

Articles in the NB Series are designed to accommodate information at three independent but complementary levels (or "tiers"), which are shown in the following table:

Tier	Description
Core Architecture	High-level, architectural information; "schematic-level " view of functionality; provides contextual overview/baseline knowledge
Conceptual Application	Best practices and guidelines associated with CRM features or functionality that can be applied based on the specifics of particular implementation
Practical Application	Detailed explanations about how to address unique scenarios; practical details about resolving issues or accomplishing specific "real-world" tasks

This component of the Nuts and Bolts article *Security and Authentication in Microsoft Dynamics CRM* addresses selected aspects of the conceptual application of the Microsoft Dynamics CRM 4.0 security model.

The full breadth of coverage provided by the Nuts and Bolts article *Security and Authentication in Microsoft Dynamics CRM* includes the following:

- Core Architecture
  - *The Microsoft Dynamics CRM Security Model*
- Conceptual Application
  - *Securing Microsoft Dynamics CRM in the Enterprise*
  - *Field-level Security in Microsoft Dynamics CRM: Options and Constraints*
  - *Securing the Network Infrastructure for Microsoft Dynamics CRM*
- Practical Application
  - *Security Contexts in Microsoft Dynamics CRM*
  - *Connectivity and Firewall Port Requirements in On-Premise Deployments*

Introduction

Many businesses have sensitive data that should only be viewable or editable only by certain groups of users. While Microsoft Dynamics CRM offers the ability to limit access at the entity level, business requirements frequently include controlling access to and updating data at a more granular level.

This business need is typically voiced through a customer request for field-level security, which would provide the ability to:

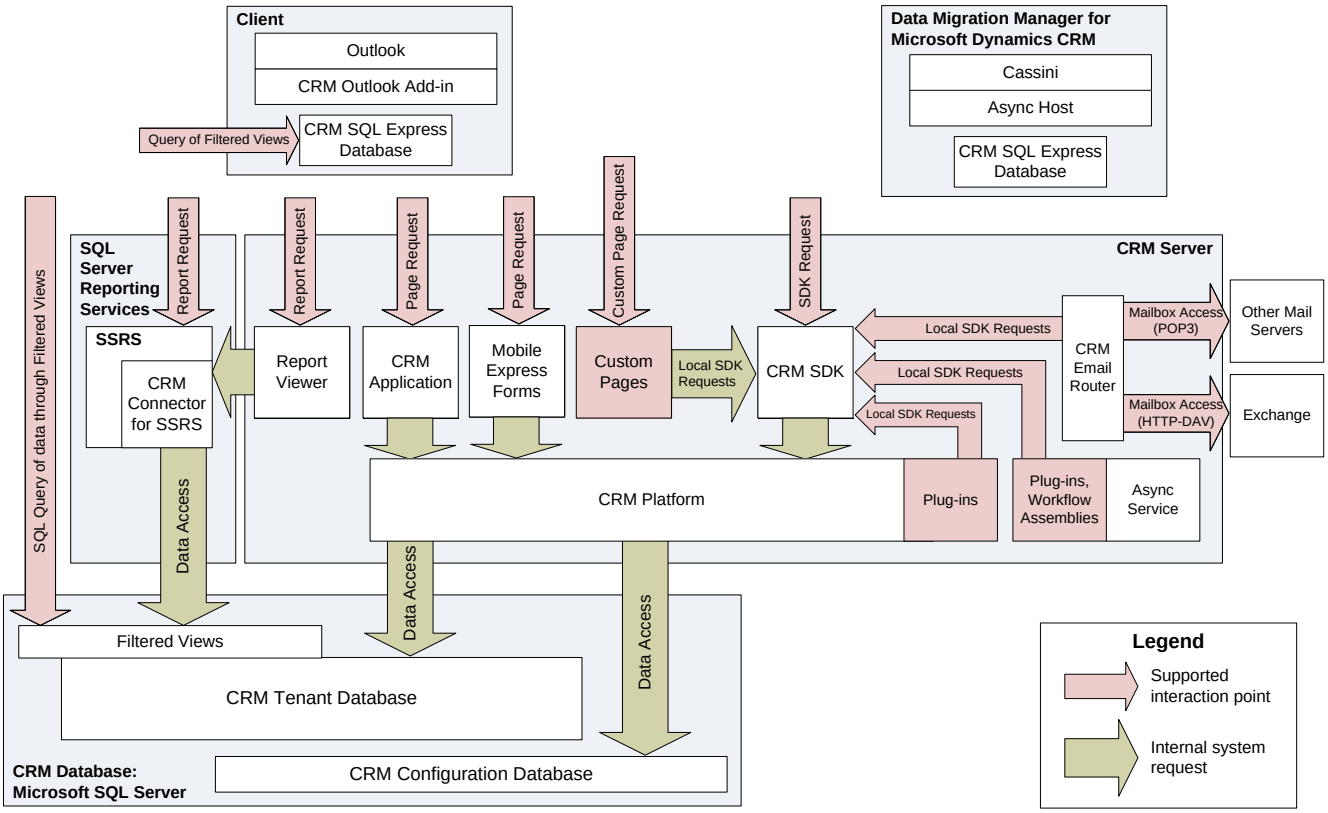
- Hide certain fields from particular groups of users
- Allow particular groups of users to view but not edit certain fields of an entity

While these capabilities are not provided directly by Microsoft Dynamics CRM 4.0, there are a number of options available for using supported custom logic to control of access to data at a more granular level. This document discusses some of the options available.

Controlling Access to the Dynamics CRM Platform

Interaction Points

Microsoft Dynamics CRM provides several *interaction points*, or access methods and channels, that external users and systems can leverage to interact with the CRM platform. The following diagram shows the key interaction points associated with Microsoft Dynamics CRM 4.0.



Microsoft Dynamics CRM 4.0 does not currently provide a feature that ensures a consistent approach to providing field-level control across all interaction points in a CRM deployment. As a result, it is important to consider each of the potential access channels independently, focusing on the specific options that are available for intercepting or affecting the requests that are made through that channel.

Access Channels and Methods

While users and external systems interact with the CRM platform in many ways, in reality the access occurs through a set of common access channels, each of which is associated with one or more access methods.

The access channels and associated access methods available in Microsoft Dynamics CRM 4.0 are described in the following table.

Access Channel	Access Method	Description
CRM Application	CRM Web Client	A browser interface that enables users to interact with the CRM application, or with the CRM Report Proxy when generating reports.
	CRM Outlook Client (online and offline)	An Outlook-integrated interface that enables end users to interact with the CRM application, or with the CRM Report Proxy when generating reports.
	CRM E-Mail Router	The E-mail Router is an interface between the Microsoft Dynamics CRM system and one or more Microsoft Exchange or POP3 servers for incoming e-mail, and one or more SMTP servers for outgoing e-mail. E-mail messages come into the Microsoft Dynamics CRM system through the E-mail Router.
	Mobile Express	A set of web pages hosted in a sub-directory of the CRM Application targeted at a mobile browser client
CRM SDK Web Service	Plug-ins, workflow assemblies	Plug-ins and workflow assemblies can make requests of the CRM platform directly by using the SDK, for example to get data from another system and update CRM when an entity instance is created or updated.
	External applications	External applications can make requests of the CRM platform directly by using the SDK, for example to surface data from or push new data into the CRM system.
	Custom pages	Certain implementations require developers to use custom pages in the ISV web folder to make SDK calls to modify or retrieve data. **Note**: For more information about custom pages accessing the CRM SDK Web Service, see the white paper *Security Contexts in Microsoft Dynamics CRM*.

Access Channel	Access Method	Description
Filtered Views (CRM Database)	External applications	External applications may require access to the CRM database to perform bulk data retrieval; typical scenarios include: ▪ Taking advantage of database access as an integration mechanism (e.g. SQL Server Integration Services), which would be more difficult to accomplish by using the CRM SDK Web services ▪ Requiring access to volumes of CRM data, e.g. when using SQL Server Reporting Services to display a CRM report **Important**: Accessing the tables underlying the CRM database is not supported.
	CRM Export-to-Excel feature	A feature that enables end users to export CRM data to Microsoft Office Excel. The functionality involves the CRM system downloading a worksheet and then Microsoft Excel connecting directly to the database to retrieve the necessary data by using the Filtered Views that are exposed by CRM.

This document addresses each interaction point and the associated options that are available for restricting access to data at a granular level. Always remember, however, that the key to providing a fully secure CRM implementation is to properly secure each of the interaction points that is associated with that implementation.

Note: In some implementations, one possible option may be to block access to selected interaction points altogether, rather than providing granular access through them. For example, consider a business scenario in which:

- Managers can view all data rather than needing it filtered at a field level
- Only managers require Excel Export or Reports capability

In this case, removing the Excel Export and Reports privileges from everyone but managers would maintain secure access while avoiding the need to filter these interaction channels.

Usability Considerations

While techniques in this document can help to ensure that data is only viewable or editable by authorized users, these techniques do not account for the potential usability consequences of making data unavailable only to selected users.

For example, users can find it confusing to work with an interface that includes fields that they can edit but that might contain already data that the users cannot view. From the user perspective, the absence of data in any given field could indicate either an empty field or alternately a field containing data that the user was not authorized to view. Ambiguity in a user interface is far from ideal and should be weighed in context with the overall solution being implemented.

While the appropriate way to address usability considerations depends on the specifics of any implementation, the options include:

- *Passing an indicator to the field to display to the user that they user cannot view the attribute e.g. '******', 'Restricted', 'Not Authorized'*.

  Not all field types lend themselves to display of a restricted indicator. For example a check box which simply has a checked or unchecked state does not allow for the display of an indicator that the information has been restricted.

- *Making fields read only or hidden if a user is not authorized to view them.*

  This could be triggered on script that detects the indicator used, as suggested above, to suggest to users that the field is restricted. Alternatively, a separate mechanism could be used, such as a hidden attribute, to identify to the client which fields should be hidden.

Important: From a design perspective, if values for an attribute are hidden from a user or replaced with alternative text, then the corresponding Update and Create Plug-Ins should be provided to prevent accidentally overwriting the real data. Additional information about implementing these plug-ins is provided in the section "Restricting Access by Using Plug-Ins."

Role- and Rule-Based Filtering

Another important part of designing field-level security for a CRM implementation is to define which users have access to which fields and the basis upon which users are allowed or denied access to that information.

The type of information that is required to define whether or not a user gets access to a particular field can affect the choice of an appropriate mechanism to provide that control.

Field-level security is most commonly defined as either *role-based* access or *rule-based* access. The choice between role and rule based filtering is an important one, with tradeoffs between complexity, performance and flexibility, as explained below.

Role-Based Filtering

This is the simplest and most common approach to defining field-level security. With this approach, access to attributes of an entity is defined for each user role. If the user does not have a role that is entitled to access the attribute then it will be restricted.

An advantage of role based filtering (and a reason for its simplicity) is that the access to an entity's attributes can be defined without needing to access the data in the entity's records, and that the definition will apply uniformly across the instances of an entity for a particular

user. This enables filtering of the attributes to be retrieved from the CRM platform before the query is run, which can therefore reduce the amount of the data requested from the CRM platform reducing the performance overhead of the query.

Rule-Based Filtering

Some cases require a more complex access scheme, which can involve specifying a series of rules to define the conditions under which an entity's attributes can be accessed. These rules could require access to and comparison of information about a user or about the attributes of:

- Each record of the entity being accessed
- Entities related to the entity being accessed

The complexity and definition of the rules may allow for building extra rules into the query that is used to request information from the CRM platform. Alternatively, it may be necessary to filter after retrieval, though this results in the additional performance overhead associated with having retrieved data that will be filtered out before information is displayed to the user.

As for role-based filtering, if the rule set defines a set of attributes that will be hidden for all the records that can be retrieved, for example because the rule is based on user attributes only, then the filtering can be applied before the query is retrieved and the columns can be removed from the original query.

If attribute filtering depends on the data within the entity records or in related records and the filtering can vary for each record, then retrieving all the records and applying the filtering to each individual record in the result set is required. Because the query language only allows query-level specification of the attribute set to be returned rather than providing record-level rules, the same set of attributes will be returned for each record that is returned. As a result, record-specific filtering must be performed after retrieval.

Consider a scenario in which business rules require that all records with a customer turnover:

- Greater than or equal to £1M have the profit field restricted
- Less than £1M have the profit field displayed

Because the platform cannot return the profit for one subset of the customers but hide it for the others, it is necessary to query the platform to return the profit for all customer records and then to filter out the values of the returned records set afterwards before returning it for display at the client.

Options for Filtering Data

There are several options available for restricting user and system access to data in a Microsoft Dynamics CRM 4.0 implementation, and those options vary depending on the nature of the interaction point to be secured.

Note: A summary of the mechanisms available for providing field-level security in a Microsoft Dynamics CRM 4.0 implementation appears in *Appendix A: Summary of Filtering Mechanisms*.

Access via Forms

As the primary mechanism for displaying entity data, CRM Entity forms are a critical component to secure.

Important: Often, it is recommended to use a hybrid approach, leveraging multiple means of restricting access to the data in forms, to ensure the best possible solution.

Consequences of Restricting Access by Using Client-Side JavaScript

A common design misconception about providing field-level security in CRM forms is that client-side JavaScript can be used to secure the access. In fact, client-side JavaScript does not provide true field-level security access because although a field may appear inaccessible to a user, the field data is still transmitted from the server to the client and a number of techniques could be used to retrieve the information at the client or at the network level.

Similarly, because attributes of a record can be set as part of the create process by passing attributes on a manually generated query string directly to the server, relying on client side scripting to provide field-level security allows for a user to set values for fields on new records that are supposedly secure fields that they should not be authorized to access.

While the client-side JavaScript does not provide true security, it is often beneficial to use this technique it to enhance usability by providing users with immediate feedback regarding the field restrictions. As a result, client-side JavaScript is frequently an important component in establishing a Field Level Security implementation, particularly within Forms. However, it is equally important to realize the primary purpose and benefit of using client-side JavaScript is to enhance User Experience and usability rather than to provide the underlying security.

Restricting Access by Using Plug-Ins

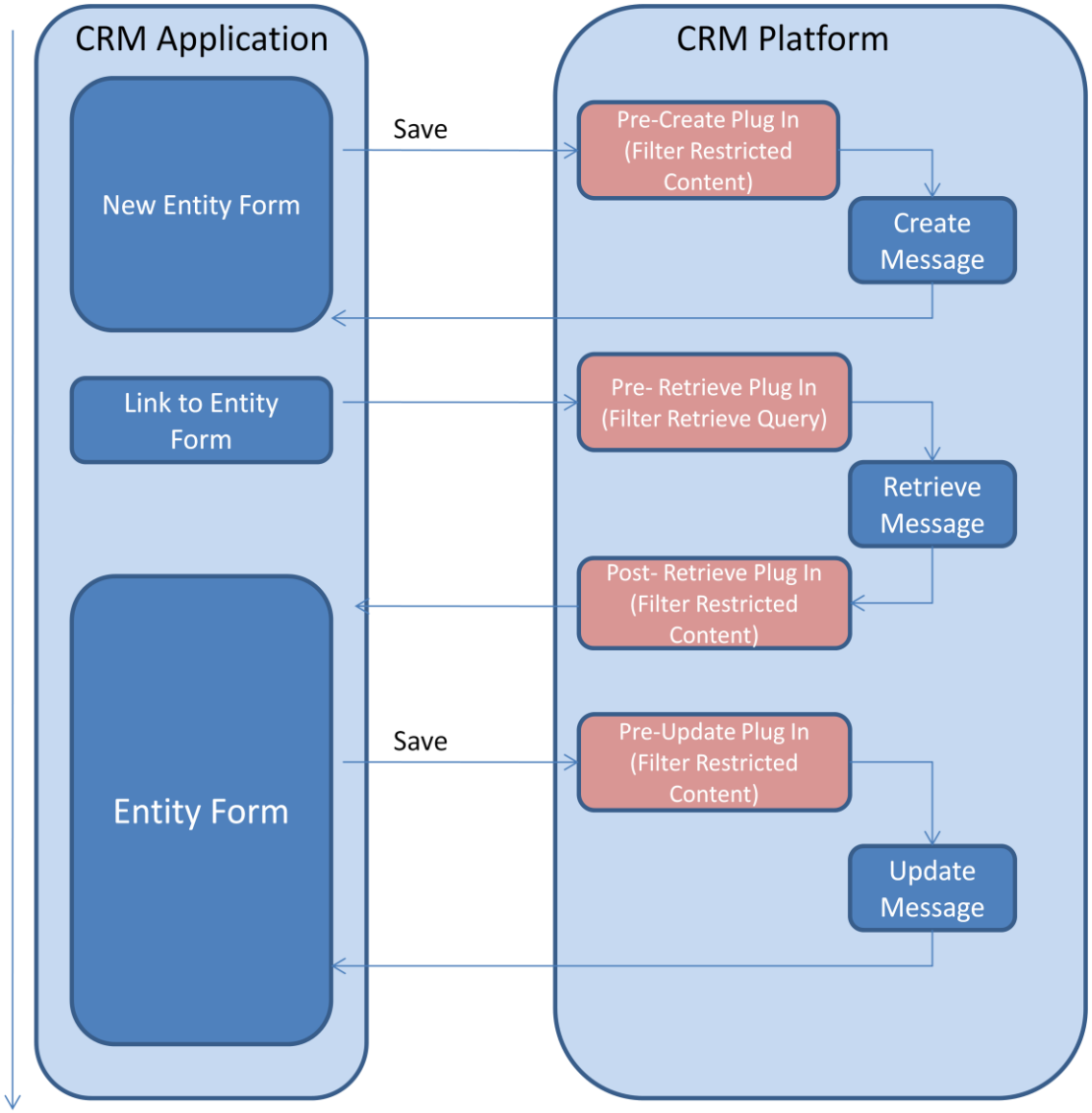
By implementing entity plug-ins that provide the logic to restrict access, it is possible to provide field-level security access to CRM forms. However, it is important that the restrictions be performed at the server level to prevent the mechanisms from being bypassed by a technically capable, unauthorized user.

Scenarios that require restricted access to data must account for the different ways that forms are used (Create, Retrieval, and Update). During the creation of a record, for example, it is important to prevent accidental or malicious entry of unauthorized data. This can be accomplished by implementing the appropriate logic within a Pre-Create Plug-In.

During Retrieval, it is important to hide data from unauthorized users. This can be accomplished by using either a Pre-Retrieve or Post-Retrieve Plug-In, in each case preventing the retrieval and display of the restricted data within the platform. This avoids exposing the data to any client-side process.

When data is hidden from users as part of the form display, it is important to prevent the platform from accidentally overwriting actual data with blanks or some form of a 'Restricted' indicator. As a result, it is also necessary to provide a Pre-Update Plug-In to prevent unauthorized users from updating record fields that those users are not authorized to view or to update. This logic can also be used to prevent accidental or malicious updates to restricted fields by query strings.

The overall flow of interactions between the user interface at the client and the underlying CRM platform and how the interactions typically occur is shown in the following diagram:



By using this combination of plug-ins, it is possible to provide a wide range of flexibility, including but not limited to:

- Field-level, read-only behavior
- Full restriction, so that a user can neither view nor update a field.

The logic defining what access restrictions should be performed can be based on either role- or rule-based filtering.

Access via Views/Advanced Find

The CRM user interface includes a number of options for searching entity data, and each option returns a list of records, such as entity views and advanced find. You can provide Plug-Ins to intercept and filter these recordsets at the platform level, which would ensure that secure data is not accessible to unauthorized users through these mechanisms.

The underlying queries that these user interface elements use to retrieve data are sent to the platform via one of two types of messages, either by FetchXML requests sent through the Execute message or by RetrieveMultiple messages. Both types of messages must be filtered to ensure data security.

It is possible to filter these retrieval requests either before or after running the platform query. Filtering the columns to retrieve in the 'Pre' step retrieves less data in the first place, which benefits performance. On the other hand, if context-sensitive filtering is required, for example to block the display clients with more than £100K worth of business with the company, then this would have to occur in the 'Post' step, after the client volume data had been retrieved. Each approach is valid, used either in isolation or in a combination; the appropriate solution with depend on the business context of the filtering required.

Limiting access to the filter and order criteria may also be required to prevent users from deriving the secure data through careful searching. For example, a user might produce advanced find filter criteria to determine all the customers with a 'Turnover' attribute value of £1m, even though the 'Turnover' attribute may not be returned in the result set. Fully securing this data would also require preventing users from filtering on the secure attributes.

Access via the CRM SDK Web Service

Access to data by using the CRM SDK Web service is also important to control, and as with controlling access to data from the User Interface, using Plug-Ins can provide an appropriate solution. For each entity that must be restricted, follow the patterns described for restricting access to data through the User Interface to implement the following Plug-Ins:

- Retrieve
- Create
- Update
- RetrieveMultiple
- Execute : FetchXML

Access via the CRM Metadata Web Service

The CRM Metadata service provides access to the definitions of the entities and their attributes for a particular CRM implementation. This enables making a request to query for the existence of certain attributes on an entity.

While it may be desirable in certain circumstances to use field-level security to hide the existence of an attribute from a particular user who does not have access to that attribute, it is not currently be possible to customize the results returned from the metadata service. As a result, it would not be possible to hide the fact that information of a certain type is captured for a particular entity, even though by using the other mechanisms it would be possible to hide the actual data captured against that attribute.

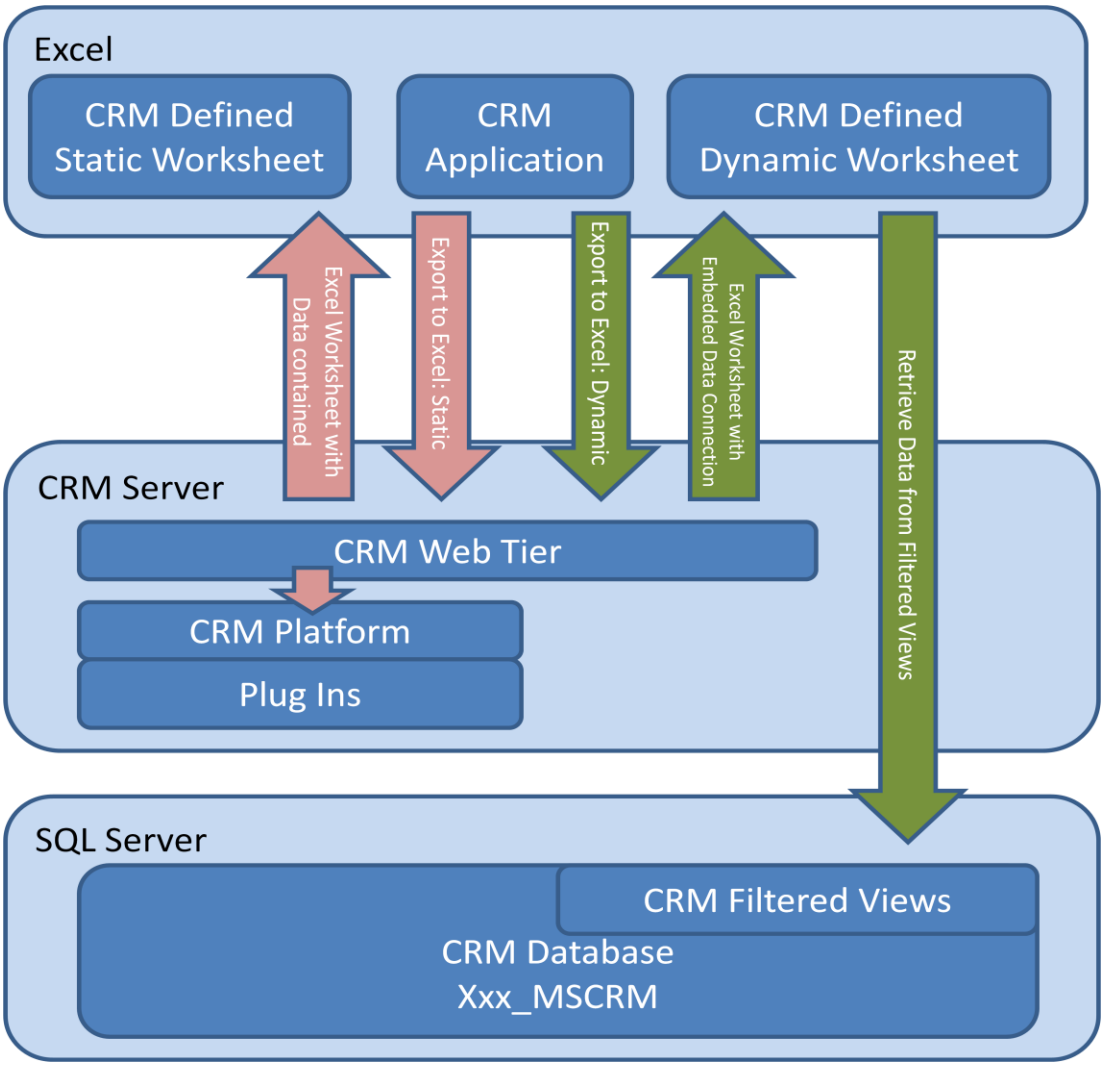
Access via the Export-to-Excel Feature

Microsoft Dynamics CRM 4.0 provides an Export-to-Excel feature, which allows users to download an XLS file that can either:

- Contain CRM data as static information in the worksheet
- Dynamically retrieve the data from CRM through an embedded data connection

For normal entity level security management as offered by the native CRM product, the Export-to-Excel feature ensures that users can only export the data to which they have access. After enabling field-level security by using the plug-in approach described above, it is important to consider the Export-to-Excel capability to prevent data restricted through the UI from being accessed in this way.

The Export-to-Excel feature functions in fundamentally different ways depending on whether a user requests a Static worksheet or a Dynamic PivotTable/Worksheet, as is shown in the following diagram:



With static worksheets, the data is retrieved within the platform and returned as part of the resulting worksheet. As a result, the data can be filtered before it is downloaded to the client Excel worksheet by using the FetchXml and RetrieveMultiple messages described previously.

Applying field-level security is more problematic with dynamic worksheets however, because the data is retrieved directly by the Excel worksheet at the client from the database by using the filtered views that the CRM product creates and exposes. There is no supported way to intercept or alter the data returned by the CRM provided Filtered Views. As a result, the best option available is to prevent direct access to the database Filtered Views altogether by the user from the client. While users lose the functionality provided by the dynamic worksheet feature, this would require users to access data by using channels to which field-level security mechanisms can be applied.

The options available for securing access via the Export-to-Excel Export feature therefore require access to the Filtered Views in the CRM database to be blocked.

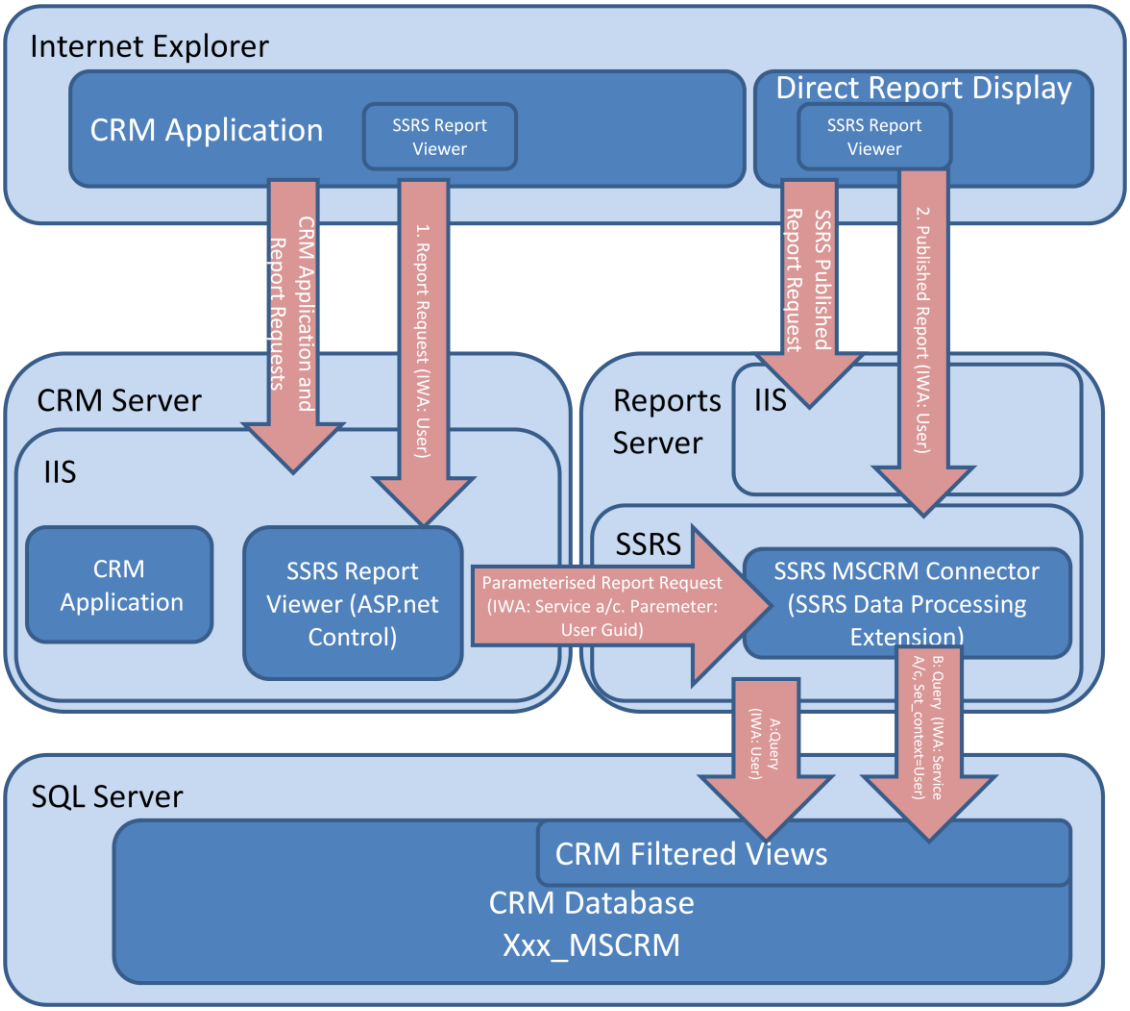
Note: Additional information about the mechanisms used to achieve this are covered in the section "Direct Access to the Database."

To secure data that can be accessed by the Export-to-Excel feature, available options include:

- **Disabling Excel Export for roles that require field-level security**. This can be performed using standard CRM role privileges
- **Allowing only Static Worksheets to be exported**. While there are no standard product implementations for this, existing options include:
  - Using a product such as Microsoft Intelligent Application Gateway (IAG) to filter requests to the CRM server and intercept worksheet files returned that contain embedded data connections will make this restriction clear to the user. A request containing a dynamic worksheet could be detected in IAG and redirected to a custom page indicated that this is a restricted action which this user is not authorized to perform. This does not prevent a user creating their own Excel Worksheet with a data connection to the CRM database and as a result, the access to the Filtered Views needs also to be separately blocked.
  - Providing a HTTP Module to block requests for dynamic worksheets

Access via Reports

While users can initiate reports by using the CRM user interface, the reports are actually rendered by SQL Server Reporting Services, as shown in the following diagram:



To secure data that can be accessed by using reports, available options include:

- **Disabling Reports access for roles that require field-level security**. Perform this by using standard CRM role privileges; it still allows users who do not require field-level security to provide their own reports (typically used for manager/more privileged users)
- **Disabling report creation for roles that require field-level security. Provide pre-defined custom reports for those users**. Prevent users who need field-level security from gaining access to the secure information by defining their own reports showing this information. Options include either producing:
  - Reports either using CRM UI or through SSRS tools that do not expose fields that need field-level restrictions
  - Custom SSRS reports (not through the CRM user interface) that implement the field-level security in the data retrieval query to the filtered views in the report.

Direct Access to the Database

In Microsoft Dynamics CRM 4.0, direct access to the database is limited to automatically generated "Filtered Views," which maintain security access at an entity level.

Filtered Views support access to data from Excel and from Reports but are also often used to provide integration with other systems. As mentioned previously, preventing direct access to the Database Filtered Views is an important step in ensuring that users cannot bypass field-level security filters by using means such as the Export-to-Excel feature. Locking down access directly to the database views can be accomplished by using infrastructure security, such as IPSec or firewall configuration, that prevents direct access from the client.

To secure direct access to Database Filtered Views, available options include:

- **Denying user membership to the CRM Reporting Group**. CRM enables permissions on the Filtered Views through membership of the CRM Domain Group 'ReportingGroup'. Removing a user from the Reporting Group prevents that user's access to the Filtered Views. CRM can automatically manage membership of this group, or an administrator can manage the membership. By managing membership manually, it would be possible to grant this membership only to users who do not require field-level security.

  Employing this approach, however, prevents user access to:

  - Any entity's data from Reports (unless the CRM SSRS Data Connector is used)
  - Dynamic worksheets via the Export to Excel feature
  - Any integration to the Filtered Views that may have been performed under their account

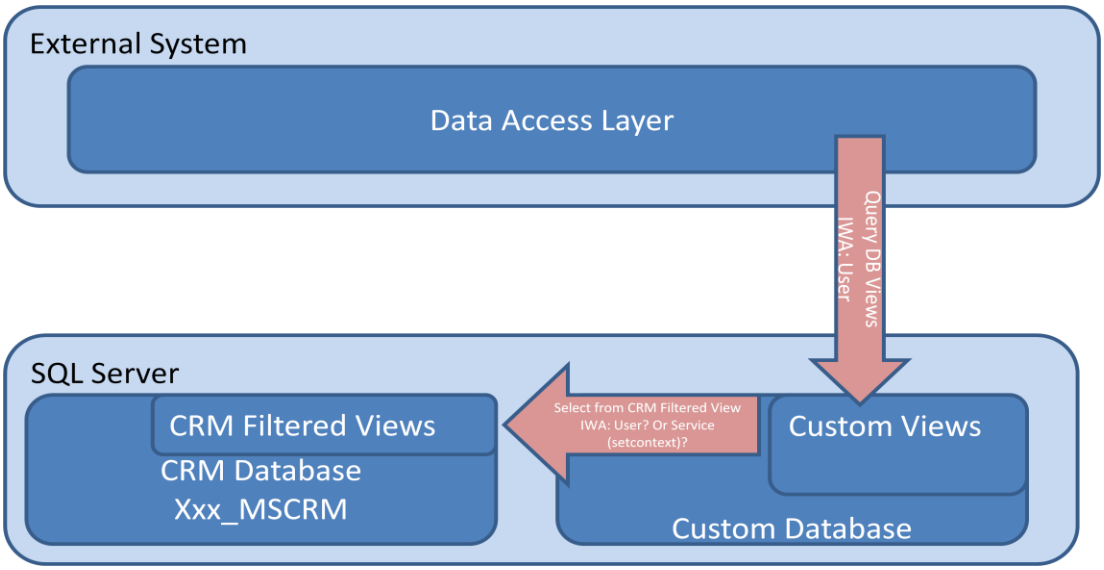
  Note: There is no supported way to limit access to specific entities.

- **Preventing access to the SQL Server database from outside of the data center**. While Microsoft Dynamics CRM 4.0 does not provide this functionality "out of the box," existing options include using:
  - Network Transport Security to prevent direct SQL Server access from the client. Mechanisms to implement network transport security include, but are not limited to, IPSec and firewall rules
  - A product such as Microsoft Intelligent Application Gateway (IAG) to filter requests to the CRM database server which this user is not authorized to perform.

Important: Each of the options available for securing direct access to Database Filtered Views rely on use of the CRM SSRS Data Connector to ensure that reports published by CRM either within CRM or externally to the SSRS web site are viewable. This is because access to the filtered views would no longer be performed under the user account but under the account running the SSRS service instead with the user identity being passed to the view as a parameter for filtering purposes.

The standard product CRM filtered views limit user access to data based upon security roles and business unit within CRM, without taking into account any additional field-level security measures that might be required for a solution.

Extending or overriding the behavior of these filtered views directly is not supported. However, for access from an external system, it may be possible to offer a separate façade in a separate custom database. This façade can provide this filtering and then be used by the external system for access, as is shown in the following diagram.



Important: If this façade approach is used but users can still access the standard CRM Filtered Views then the users can simply avoid these more restrictive views and gain full access to the data directly, making this additional layer of more limited access pointless.

Access via Outlook Synchronization

Microsoft Dynamics CRM 4.0 currently does not provide for adding field-level security to the synchronization of data between Outlook's native data storage and CRM. As a result, available options are limited. If any of the data that is shared with Outlook (i.e. contacts, appointments, tasks) needs to be limited at a field level for particular users, then those users must not be able to use the CRM Outlook add-in.

If the field-level security requirements apply only to Entities that are held within CRM and not synchronized with the native Outlook Data Storage, then the mechanisms described in previous sections should, for each CRM access channel, also apply.

Access via the Offline Client

With Microsoft Dynamics CRM for Microsoft Office Outlook with Offline Access (the "offline client"), offline synchronization follows a process similar to that for CRM-Outlook synchronization. However, computers running the offline client include a local SQL Express database, which provides for the standard CRM Filtered Views that expose data regardless of any of the field-level security filtering mechanisms described previously. As a result, limit users who require field-level security to installing Microsoft Dynamics CRM for Microsoft Office Outlook (the "online client").

Implementing Data Filters

As described, providing field-level security requires filtering data at a number of points in the system and implementation to account for a variety of different methods of access. The following sections provide details about how to implement filtering mechanisms for each method of access.

Plug-Ins

One of the primary ways to restrict access to data in a more granular way is by using Plug-Ins. Plug-Ins are available for a wide range of CRM events, which provides a lot of flexibility in controlling access to CRM data. The following sections describe Plug-Ins that are typically required to provide field-level security in an implementation of Microsoft Dynamics CRM 4.0.

Create

To prevent unauthorized insertion of information during record creation, a Pre-Create Plug-In can be used to block both forms-based and query string-based requests.

Within the Pre-Create Plug-In, any attributes to be initially set on the creation of the record will be passed in the InputParameters collection as part of the context variable. To prevent a particular user from setting specific attributes, remove those attributes from the InputParameters collection.

To remove attributes from the InputParameters collection, use code similar to the following:

```
DynamicEntity updateValues =
(DynamicEntity)context.InputParameters.Properties["Target"];
If( ([USER NOT AUTHORISED] ) &&
updateValues.Properties.Contains["ATTRIBUTE_NAME"])
{
      updateValues.Properties.Remove["ATTRIBUTE_NAME"]);
}
```

Note: The logic that is used to determine whether or not a user is allowed to set an attribute is not included in this example because that logic will project specific. If you remove fields that are required for creating the entity, the SDK call may not complete. For example "productnumber" is required when creating an instance of Product entity.

Retrieve

The Retrieve event is called for Form display and can be called independently from the CRM SDK Web service.

There are a number of options for limiting retrieval of restricted information, whether by preventing the information from being retrieved in the first place or by filtering it out after retrieval but before passing it from the server to the client. There are advantages and disadvantages associated with each approach, so determining the most appropriate method for any specific environment will be scenario-specific.

With each approach, the form itself is unaware of fields with real data hidden or overridden in some way, such as by the text 'Restricted'. As a result, there is a potential for overwriting the real data on Save from the form. Therefore, be sure to provide for a pre-update Plug-In to avoid the potential of data accidentally being overwriting by users with restricted access.

Pre-Retrieve Plug-In

When a Form is retrieved, a Retrieve event is fired. In the Request, the fields to be retrieved are defined in the InputParameters of the request. The Pre-Retrieve Plug-In fires before this query is passed to the platform for processing.

By adding logic to the pre-retrieve Plug-In, it is possible to remove from the retrieval request any fields that a user is not authorized to view.

For fields that a user is not authorized to view, remove them from the retrieval request by adding code similar to the following:

```
Microsoft.Crm.Sdk.Query.ColumnSet RetrieveColumns =

(Microsoft.Crm.Sdk.Query.ColumnSet)
context.InputParameters.Properties["ColumnSet"];

RetrieveColumns.RemoveColumn("fax");
```

Adding such logic prevents the retrieval of values associated with the columns removed, and no data leakage can occur. However, this solution does not allow for flexibility in restriction based on information included in the record.

Post-Retrieve Plug-In

After the platform has performed a query but before the data is returned to the client, the Post-Retrieve Plug-In will fire, and at this point the resulting information about the record will be contained in the OutputParameters. If a Pre-Retrieve Plug-In is used to remove fields from the retrieval query, then nothing needs to be performed by the Post-Retrieve Plug-In.

However, by adding code to the post-retrieve Plug-In, it is possible to restrict access to fields in a record based on other information about the record. For example, a business might decide that only Sales Managers or the owners of a record should see the sales volumes for VIP customers. In this scenario, filtering the data in the Post-Retrieve Plug-In would be more appropriate that it would be in the Pre-Retrieve Plug-In.

To restrict access to fields in a record based on other information about the record, use code similar to the following:

```
 DynamicEntity outputValues =
(DynamicEntity)context.OutputParameters.Properties["BusinessEntity"];

 outputValues.Properties["salesvolume"] = "Restricted"; // Or " "
```

Update

To prevent unauthorized updates to fields in a record, add a Pre-Update Plug-In that can filter out any changes that users may make but that they are not authorized to perform. This can also be used to prevent accidental overwriting of values that were hidden during retrieval.

When an update occurs, the fields to be updated are passed in the InputParameters collection. Fields that are not being updated will not be present. If a user is not authorized to view or update certain fields, then in the Pre-Update Plug-In, the InputParameters collection can be checked for the existence of updates to these fields and they can be removed from the collection, thereby preventing the update.

This solution can also be used to provide field-level update permissions, so that a user can view a record's fields but not update specific individual fields.

To create a Pre-Update Plug-In that can filter out any changes that users may make but that they are not authorized to perform, use code similar to the following:

```
DynamicEntity updateValues =
(DynamicEntity)context.InputParameters.Properties["Target"];

If( updateValues.Properties.Contains["ATTRIBUTE_NAME"])
{

updateValues.Properties.Remove["ATTRIBUTE_NAME"]);

}
```

Execute: FetchXML

Most system views query the platform for data by using a FetchXML request through the Execute message. Providing a Plug-In on the Execute message allows for capturing these events and filtering of either the:

- Query being performed on the Pre-Execute event
- Returned data on the Post-Execute event

As for filtering of Retrieve messages, the same options exist for applying the filtering logic in either the Pre-Execute or Post-Execute message. As for the Retrieve, there are advantages and disadvantages to each option, and the most appropriate should be chosen based upon the specifics of each project.

Pre-Execute

The Pre-Execute Plug-In can be used to prevent the retrieval of attributes for an entity. The main advantages of this approach are that it is relatively simple to implement and it avoids retrieving selected information, which provides performance benefits. The biggest potential disadvantage is that field filtering applies to all records returned. While this might be perfectly acceptable for scenarios in which field-level filtering is to be performed based on user role, for example, this method does not support filtering based on the particular characteristics of the record itself, for example, the value of the account, if that information wasn't retrieved. In this case, use a Post-Execute Plug-In should be used instead.

Because the Execute message is used for many purposes, upon interception of a Pre-Execute event, it is important first to determine whether or not the event is a FetchXml event. To achieve this, the Plug-In should check for the existence of an InputParameter named 'FetchXml'; if this does not exist then the event can be ignored.

If the message is a FetchXml request, then the value of the FetchXml parameter can be checked and manipulated. It is also important to determine whether or not the message is for an entity type that needs to be filtered. This needs to be determined from the XML defining the query, as the Execute method will not have the PrimaryEntityName value set.

The format of the FetchXML parameter value would be similar to the following:

```
<fetch distinct="false" mapping="logical" page="1" count="50"><entity
name="contact"><attribute name="fullname" /><attribute
```

```
name="parentcustomerid" /><attribute name="telephone1" /><attribute
name="emailaddress1" /><attribute name="contactid" /><attribute
name="fullname" /><attribute name="emailaddress1" /><attribute
name="parentcustomerid" /><attribute name="telephone1" /><filter
type="and"><condition attribute="ownerid" operator="eq-userid" /><condition
attribute="statecode" operator="eq" value="0" /></filter><order
attribute="fullname" descending="false" /></entity></fetch>
```

The code would need to:

1. Check the entity element to determine whether or not this query is for the entity to be filtered
2. Remove the XML element describing any attributes that should not be retrieved and displayed to the user

Note: There are many approaches to manipulating XML, and the appropriate method will depend on the requirements of a particular implementation. These approaches are outside the scope of this paper.

It may also be necessary to prevent users from deriving restricted information by searching or filtering on restricted field values despite the fact that the actual values for those fields are never displayed to those users. This might occur for example if a user was not permitted to view the customer volume of business field but was able to do a search for customers with a volume of business greater than or less than a certain amount.

If this type of manipulation might lead to data loss, then preventing searches on the restricted attribute would also be required. The level of protection required will be very implementation specific, but a simple recommendation would be to prevent filtering or ordering by a field that is hidden or restricted for the requesting user.

To prevent this, it would be necessary to check for any use of the attribute within the query, i.e. any reference to the attribute in the filter, condition, order, or link-entity elements of the query. If the attribute is referenced, then manipulation of the remaining query could be both complex and misleading, as it may not provide the expected results without the extra search clause. Instead, it is recommended that the query be rejected, an error be reported to the user, and the user be left to resolve it.

Post-Execute

If the filtering method requires knowledge of information about a client, for example the value of the account, then the filtering should occur in the Post-Execute Plug-In.

Intercepting the Post-Execute event enables more granular filtering of data per record than can be performed in the Pre-Execute plug in. Upon interception of a Post-Execute event, it is important first to determine whether or not the event is a FetchXml event. To achieve this, the Plug-In should check for the existence of an OutputParameter named 'FetchXml'; if this does not exist then the event can be ignored.

For FetchXML requests, it is next important to identify if it is for an entity type that needs field-level filtering. Although the entity type is not present in the result set, the query details will still be available in the InputParameters request and the entity element can be checked as for the Pre-Execute Plug-In approach.

The value returned in the FetchXmlResult Output Parameter will look similar to the following:


```
<resultset morerecords="0" paging-cookie="&lt;cookie
page=&quot;1&quot;&gt;&lt;fullname last=&quot;Test Create&quot;
first=&quot;Hello World&quot; /&gt;&lt;contactid last=&quot;{E0506806-AB18-
DE11-BB7C-0003FFCFA222}&quot; first=&quot;{A0F489CA-E4DF-DD11-BD6E-
0003FFCFA222}&quot; /&gt;&lt;/cookie&gt;"><result><fullname>Hello
World</fullname><contactid>{A0F489CA-E4DF-DD11-BD6E-
0003FFCFA222}</contactid></result><result><fullname>Joe
Bloggs</fullname><contactid>{C01763FA-6DE1-DD11-8B7E-
0003FFCFA222}</contactid></result><result><fullname>Joe
Worsley</fullname><contactid>{A0A8CC31-7DE1-DD11-8B7E-
0003FFCFA222}</contactid></result><result><fullname>John
Smith</fullname><contactid>{C0CDE50A-E6DF-DD11-BD6E-
0003FFCFA222}</contactid></result><result><fullname>Martin
Corry</fullname><contactid>{608E87A7-97E1-DD11-8B7E-
0003FFCFA222}</contactid></result><result><fullname>Martin
Johnson</fullname><contactid>{401C4BF7-81E1-DD11-8B7E-
0003FFCFA222}</contactid></result><result><fullname>Mickey
Mouse</fullname><contactid>{3099FB17-7CE1-DD11-8B7E-
0003FFCFA222}</contactid></result><result><fullname>Mike
Catt</fullname><contactid>{D01479BB-78E1-DD11-8B7E-
0003FFCFA222}</contactid></result><result><fullname>Neil
Back</fullname><contactid>{20931875-B8E1-DD11-8B7E-
0003FFCFA222}</contactid></result><result><fullname>Richard
Hill</fullname><contactid>{F0BF8075-97E1-DD11-8B7E-
0003FFCFA222}</contactid></result><result><fullname>Test
Create</fullname><contactid>{E0506806-AB18-DE11-BB7C-
0003FFCFA222}</contactid></result></resultset>
```


Any values that should be hidden should be either removed from the XML or the value overridden as appropriate for the solution.

An edge case to note is that only information returned in the query can be used to filter the results set. As a user could use Advanced Find to define the query set, it is not possible to guarantee that any particular fields will be in the query set.

This is particularly relevant in a case where the logic to determine whether a field should be hidden is based on some other attribute about the entity e.g. the value to the business of the client, if the user does not add that attribute to the query then the values won't be present to perform the calculation on.

In this case it would be possible in the Pre-Execute to ensure that the extra attribute is added to the query so it is available in the Post-Execute result set to perform the calculation. This extra set of values then could be passed on to the client and just not used or a more efficient solution could also filtering out the added attribute set the Post-Execute plug-in after it has been used to calculate the filtering of other dependent attributes.

RetrieveMultiple

Several places in the User Interface use the RetrieveMultiple message rather than FetchXml to query the platform, as does the CRM SDK Web service. As a result, the RetrieveMultiple message must also be filtered to provide field-level security. While data is exposed differently within the RetrieveMultiple message Plug-Ins, the actual underlying data is the same and the same changes can be made, although they are done by using the SDK .net wrapper classes rather than by having to manipulate XML.

Pre-RetrieveMultiple

The Pre-RetrieveMultiple Plug-In can be used to restrict the fields that are retrieved as part of the RetrieveMultiple message. As with the FetchXml request, this involves removing columns from the Query ColumnSet, but in this case the manipulation can occur directly to the SDK .Net classes:

```
Microsoft.Crm.Sdk.Query.QueryExpression currentQuery =
(Microsoft.Crm.Sdk.Query.QueryExpression)context.InputParameters.Properties[
"Query"];

Microsoft.Crm.Sdk.Query.ColumnSet currentColumns =
(Microsoft.Crm.Sdk.Query.ColumnSet)currentQuery.ColumnSet;

currentColumns.RemoveColumn("fax");
```

As for the FetchXML case, it is also important to ensure that the query being made does not include any references to the restricted attributes in the filter, condition, order, or link-entity elements of the query. If there are, the recommendation would be to reject the query with a warning to the users regarding why it was rejected.

Post-RetrieveMultiple

Also was the case with the FetchXml request, the results of the query can be processed after the data is retrieved. However, in the case of the Post-RetrieveMultiple message, this can be accomplished by using the SDK .net classes rather than by manipulating XML.

In this case the returned Business Entity classes are manipulated, as shown in the following example code:

```
BusinessEntityCollection outputValues =
(BusinessEntityCollection)context.OutputParameters.Properties["BusinessEntit
yCollection"];

foreach (DynamicEntity nextContact in outputValues.BusinessEntities)

{

nextContact.Properties["middlename"] = "Restricted";

}
```

The Export to Excel Feature

As mentioned previously, if a project requires allowing for Export-to-Excel functionality but also for field-level security on the data returned, then it is necessary to prevent access to Dynamic Worksheets.

Note: Preventing access to the data retrieved through the Filtered Views may also be desirable to prevent users from downloading dynamic worksheets that will simply fail with an access error later.

This can be performed by using an application such as Microsoft's Intelligent Application Gateway (IAG), which includes a feature that provides the ability to redirect URL requests based on a set of rules, such as user role or content of the request.

In this scenario, IAG could be used to detect requests to the URL /CRMCONFIG/_grid/print/export_live.aspx and if the user is associated with a role that is not allowed full access to the data, then the request could be redirected to a page that explains that the user is not authorized to use this feature.

If all users are to be blocked from downloading dynamic worksheets, then the URL Rewriting feature in Internet Information Server 7.0 can be used to provide the same redirection for requests made to that URL. If only certain entities are restricted from being exported to Excel, then the request or response could be parsed for a query for the entity being requested and an error provided to the user only if it is a restricted entity for that user.

It is also possible to implement logic similar to that provided by IAG by using an implementation of an HTTP module that parses the incoming request or outgoing response and writes an error message to the response object. Using this methods would also allow the use of detailed rules to determine if access is permissible.

To check the request, a rule would have to look for an 'entity' element that included an attribute called 'name' that matched the system name of an entity with restricted access. To check the response, a rule would have to look for a data connection string that contains the entity name in the query statement as shown in the example below:

```
<QuerySource><Connection>DRIVER=SQL Server;APP=Microsoft Office
2003;Network=DBMSSOCN;Trusted_Connection=Yes;SERVER=crmconfig;DATABASE=CRMCO
NFIG_MSCRM</Connection><CommandText>select top 10000 contact0.fullname as
'fullname', contact0.parentcustomeridname as 'parentcustomeridname',
contact0.telephone1 as 'telephone1', contact0.emailaddress1 as
'emailaddress1', contact0.contactid as 'contactid' from FilteredContact as
contact0 where (contact0.ownerid = dbo.fn_FindUserGuid() and
contact0.statecode = 0) order by contact0.fullname asc, contact0.contactid
asc</CommandText><VersionLastEdit>2</VersionLastEdit>
```

Performance and Scalability

While implementing the mechanisms above provide for field-level control to data access within a Microsoft Dynamics CRM 4.0 implementation, they will also have an impact on the system, both in terms of performance for individual requests but also in overall scalability.

For each request, there is overhead associated with retrieving the rules and data and then performing any filtering required. While some of this overhead can be mitigated by loading and then caching the rules, the rules must still be applied to each request.

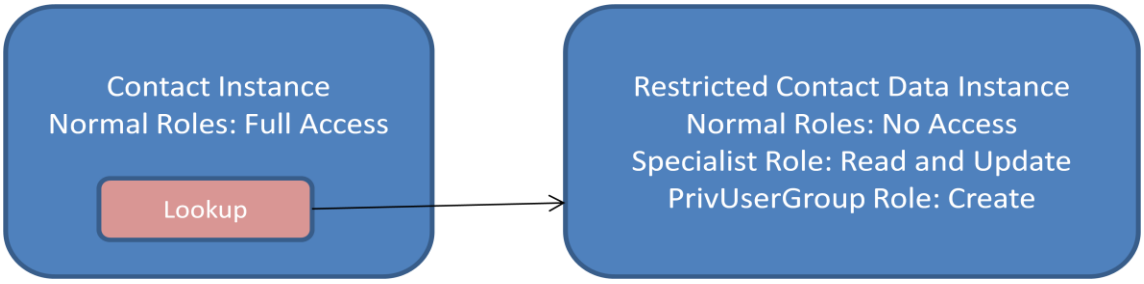
While this tradeoff may be acceptable either in slower response times or by using a higher specification infrastructure, it should be carefully considered from the initial design. In particular, it may be worth considering decisions that simplify the requirements to allow for role- rather than rule-based filtering to reduce the performance/scalability overhead.

Alternate Design Patterns

Instead of using CRM normally but offering a more granular access to data through the existing mechanisms, it may be possible to design more granular access into the solution through alternate approaches to modeling the data. This section discusses alternate designs that may be useful in addressing business requirements.

Multiple Entities

It is possible to break out data across multiple entities and then control access to the additional data based on security roles. Consider the following example, in which business requirements provide for limiting certain users' access to some details of a Contact.



This requirement could be addressed by:

1. Creating a separate entity containing the restricted access data
2. Adding an N:1 relationship with associated lookup to this new entity from the Contact entity
3. Ensuring that no existing roles have any rights to create, read, or update this new custom entity
4. Ensuring that assignments and deletes are cascaded from the contact entity to the new custom entity
5. Within the pre-create Plug-In for Contact, using the Service account (member of PrivUserGroup) to create an instance of the 'Restricted Contact Data' entity and add a reference to this new instance to the lookup on Contact
6. Creating a new role that allows read and update privileges to this new Restricted Contact Data Entity and adding this role to any users who are authorized to access the restricted contact information.

Controlling creation of instances of this entity by using the service account makes it possible to ensure that only a single instance occurs for each Contact. The restricted data will then be accessible via a normal navigation link on the Contact data Entity form, but only for users to whom the entity type is accessible.

Characteristics

Characteristics of this approach include the following:

- Works well for scenarios that require restricting selected users' access to a particular set of data across all records, or by business unit, but that is different to the group of users who need to access the parent record, for example Contact in this case.
- Does not seamlessly surface the restricted data within the contact record and requires two clicks to get to the restricted data

- Ensures that all features, such as Export-to-Excel and Reports, are fully available without any potential unauthorized access to data.
- Limits retrieving or searching for restricted data within Views and Advanced Find to authorized users.

As mentioned previously, one of this approach's flaws is the way that the additional data is displayed as a related entity, and although only a single instance occurs for each parent record, to display the sub-entity form the user is still forced to navigate to the related item link and select the single instance from the grid.

Enhancements

There are a number of possible ways to enhance this approach. For example, instead of using the automatically created related entity link on the parent entity form, it would be possible to provide a custom link directly to the related sub-entity form. This would typically be via a custom asp.net page, which would retrieve the related entity GUID and then redirect to the entity instance form. It could also be via a toolbar button or menu item in which the URL of the related entity form could be calculated in the button JavaScript.

An alternative approach would be to display the sub-entity form within an iFrame on the parent entity form. The URL for the sub-entity form can be calculated either in the onLoad JavaScript event or in a Retrieve Plug-In. If the user is able to update the information then the Save event of the parent form would need to trigger the save of the sub-entity form as well to ensure the additional data is also saved

Augment on Retrieve

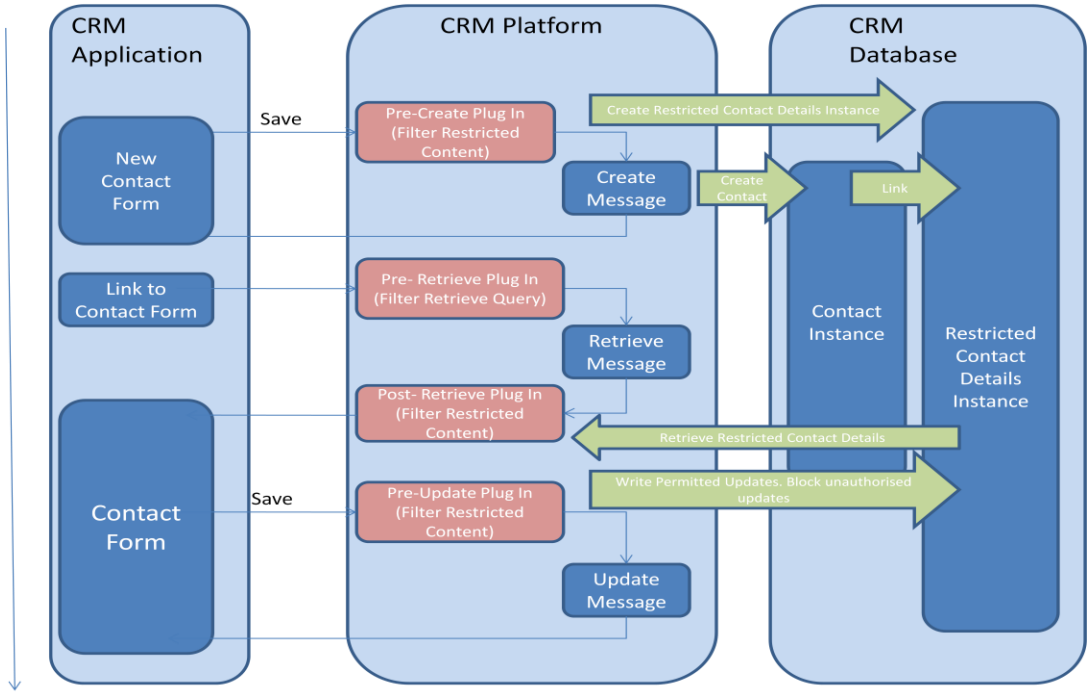
One way to ensure that users can only access appropriate data is to retain the data separately and then to augment that data to the records that are retrieved. This solution can also provide a more seamless user interface experience than would the previous approach. This approach can be used either with a separate entity as described in the previous section or with an external data source.

Similar to the Plug-In mechanisms that can be used to filter out the data that is shown in the User Interface, in the Post-Retrieve and Post-RetrieveMultiple, it is possible to include additional result sets or result attributes with data retrieved from external sources.

To the user it will appear as if the data is part of the main record, but in reality it is held separately. In this case, all the features of CRM can be safely provided to users because restricted information is not held as part of the main entity and is only exposed as the implementer chooses.

The main disadvantage to this approach is that any mechanism that relies directly on the filtered views does not retrieve this information automatically as part of the parent record. It would instead have to retrieve this information independently or use the CRM SDK Web service instead. Examples of this would include the dynamic worksheets in Excel or Reports.

The interaction between the elements making up this approach is shown in the following graphic:



If the CRM security model does not provide for the flexibility of access that is required for a particular implementation, this approach also has the additional benefit of providing for custom access. By removing all access privileges from the normal user set and only providing read and write access through the PrivUserGroup, it is possible to provide access only via custom code in the Plug-Ins.

In the Plug-Ins providing the augmented data, normally the request would be made as the account under which the CRM application pool is running, who would be a member of the PrivUserGroup. The user's GUID would be passed as a parameter to the SDK so that the request was performed on their behalf and within their security privileges.

In this case, the GUID of the service account would be passed so that any action to the restricted entity is possible and the control of access to the data would only come through the custom code provided in the plug ins. The normal CRM UI or SDK would have no access to the restricted data on behalf of the user. This enables as flexible a set of access rules to be provided for this data as required although only for access channels that trigger the Plug-Ins to perform the augmentation.

Different Channels

A slightly more complex but more flexible option is to provide different channels to users with different access requirements. Although these become more custom and more difficult to implement, there are a number of options to provide this, for example:

- Using of the eService Accelerator
- Providing a custom SharePoint Portal

Conclusion

Although Dynamics CRM does not currently offer field-level security as a product feature, this paper has shown that it is possible to use the flexibility of the CRM platform to offer relatively rich capabilities.

For scenarios that require more granular control of access to data, it is worth considering these approaches although there is no one solution that meets all needs. Rather, there are a number of possible approaches that can be combined to offer the optimum balance of capability for the requirements of a particular implementation.

Remember, though, that all possible access channels need to be considered when providing field-level security. Offering restricted access in Entity Forms and Views but allowing Reports to access the restricted data is not a valid solution unless you can secure that channel by, for example, restricting access to that channel to authorized users. In other words, it is essential to consider the whole solution when addressing these types of requirements.

Appendix A: Summary of Filtering Mechanisms

The following table provides a summary of the filtering mechanisms associated with providing field-level security in a Microsoft Dynamics CRM 4.0 implementation.

Channel	Component	Filtering Mechanism
CRM Application	Forms	• Post-Retrieve Plug-Ins • Pre-Create/Pre-Update Plug-Ins
CRM Application	Views	• Post-RetrieveMultiple Plug-Ins • Execute:FetchXML Plug-Ins
CRM Application	Advanced Find	• Post-RetrieveMultiple Plug-Ins • Execute:FetchXML Plug-Ins
Custom UI	SDK	• Post-Retrieve Plug-Ins • Pre-Create/Pre-Update Plug-Ins • Post-RetrieveMultiple Plug-Ins • Execute:FetchXML Plug-Ins
Custom Integration	SDK	• Post-Retrieve Plug-Ins • Pre-Create/Pre-Update Plug-Ins • Post-RetrieveMultiple Plug-Ins • Execute:FetchXML Plug-Ins
Reports	DB Filtered Views	• Custom reports • Role privileges
Excel Export	DB Filtered Views/ SDK	Role Privileges For Static worksheets: • Post-Retrieve Plug-Ins • Pre-Create/Pre-Update Plug-Ins • Post-RetrieveMultiple Plug-Ins • Execute:FetchXML Plug-Ins For Dynamic worksheets: • Restriction of direct access to CRM database • IAG implementation blocking worksheet download
DB Access	DB Filtered Views	• Restriction on direct access to CRM database